AIDA
Automatisering Ingest Digitale Archieven

SCALA AIP spec

Version 1.0 - 2022-04-25

Introduction

This document gives an overview of the SCALA AIP specification from the perspective of ISO 16363. We also look into ISO 16363 criteria for ingest, specifying the type of content that must be stored by SCALA's AIPs.

The document consists of three parts:

1. A section describing the AIP from the perspective of ISO 16363, offering basic information about the AIP.
2. The AIP specification.
3. A reference to AIP samples on Github.

SCALA AIP description according ISO 16363

This section describes the basic characteristics of the SCALA AIP. It explains the rationale behind the SCALA AIP specification and serves as a reference in case of further developments. The section describes the digital material that will be processed, the SIP that is ingested and the AIP that is used for storing it.

For convenience, this section makes use of ISO 16363 for explaining the SCALA AIP.[1] We use only the following parts:

- 4.1 INGEST: Acquisition of Content
- 4.2 INGEST: Creation of the AIP

1. Ingest: Acquisition of Content

1.1 The repository shall identify the Content Information and the Information Properties that the repository will preserve.

Currently, the SCALA repository is designed to process and store one type of information, what we call basic born-digital archives, originating from private archive creators.

[1] https://public.ccsds.org/Pubs/652x0m1.pdf

It is possible that SCALA will support other information types in future, but currently, these are outside scope.

The SCALA repository is designed to process and store basic born-digital archives. Basic born-digital archives are in our understanding collections of digital files:

- That are stored in a file system based on an hierarchical folder structure and filenames
- Without extra descriptive or structural metadata added, except for metadata that CAN be embedded in the files.
- That contains a wide variety of document types and file formats: text processing files, raster image files, video, CAD to even whole websites. They often contain material compressed in zip formats.
- That can show a large variation with regard to file count and file size. A basic born-digital archive can consist of three floppy disks or two 500 GB hard disks.
- That can be described according to archival methods like ISAD(G) or RiC
- That has to be processed in conformance with archival principles regarding authenticity and integrity.

These archives typically enter the collections of SCALA partner institutions years after creation, usually in a raw, relatively unordered form, e.g. after the dissolvement of a company or the retirement of an architect etc. Typically the SCALA-partners take in sets of cd’s and floppy disks, or whole hard disk drives and file servers containing a wide variety of document types stored in random directory trees.

Typically for the archival workflow a whole born-digital archive can be described according to ISAD(G) or RiC. This means the description of the archive is enriched with detailed descriptions, describing parts of the born-digital archive, mostly respecting an original arrangement present in the material. In that case a born-digital archive can be split up during archival processing in multiple IPs in relation to one detailed description. Due to practical limitations however, basic born-digital archives are not always processed this way and must be submitted to a repository without these detailed descriptions. It is a need of AIDA to be able to process the archive during a later stage after ingestion of the archive.

Basic born-digital archives are not considered to contain complex digital objects. This means one file represents one intellectual entity and is preserved as such. However, in reality basic born-digital archives CAN contain complex objects, archived websites for example. The SCALA repository will not treat these as intellectual entities in the current state of development. In order to do this, complex objects need to be ingested as a new IP category, as yet to be developed.

Content information to preserve

Files in the basic born-digital archive are minimally preserved on bit level. The SCALA repository guarantees that when they are opened by the piece of software that created them, they will be represented in the same way, given the software is configured with the same settings.

SCALA will also perform normalizations on files in the basic born-digital archive, but does not give any guarantees about the level of preservation that will result from those. Normalization operations and their objectives are described in a common preservation policy and can change over time. Changes are decided by the SCALA user group.

SCALA will keep a record about normalization policies for different file or information types on the SCALA Github account.

Information properties to preserve

- The original filepath of the files, related to their original carrier
- The files' embedded metadata at the moment of acquisition by the archive
- External references within files
- Information authenticity and integrity. Integrity will be preserved by calculating a checksum on ingestion in the repository. Authenticity is preserved by generating and preserving the necessary preservation description information. See the AIP definition for more information.
- Information useability will be preserved by storing the necessary representation information. See the AIP definition for more information.
- Information findability will be preserved by storing the necessary descriptive information. See the AIP definition for more information
- SCALA does NOT preserve the software used for creating the files in the AIP by default
- SCALA does NOT preserve the original settings of software used for creating the files by default, since this information is usually not available.

1.2 The repository shall clearly specify the information that needs to be associated with specific Content Information at the time of its deposit.

Donor agreements and all administration concerned are the responsibility of each SCALA-partner.

The SCALA repository is capable of ingesting any collection of files stored in a file system. Some metadata has to be prepared by the donor or SCALA partner in order to be added to the SIP. SCALA partners should prepare SIPs in accordance with the SCALA specification. They can do this using the tool Roda-in.

More information on the SCALA Github.

1.3 The repository shall have adequate specifications enabling recognition and parsing of the SIPs.

SCALA's SIP follows the E-ARK 2.0.4 specification as implemented by KEEP's tool RODA-in. The SCALA ingest workflow is able to parse these SIPs by help of the METS.xml files, containing descriptive and structural metadata.

During pre-ingest processing and the process of creating SIPs, SCALA partners must be aware that the file paths will change. The original path relative to the carriers should be registered in descriptive metadata at least, in order to be able to reconstruct the original situation. Empty folders will be automatically deleted.

The SCALA ingest workflow does NOT capture embedded metadata or external references in the files at the moment.

More information on the SCALA Github.

1.4 The repository shall have mechanisms to appropriately verify the identity of the Producer of all materials.

The SCALA repository only has mechanisms in place to ensure authenticity from the moment of ingest. By keeping logs and preservation description information, SCALA will at all times make sure that the provenance from the moment of ingestion can be proven.

SCALA partners have the responsibility to ensure the provenance of the material from acquisition to ingest. This can be done by storing submission agreements with lists of the material.

1.5 The repository shall have an ingest process which verifies each SIP for completeness and correctness.

The SCALA ingest workflow integrates a service (EARKSIP2ToAIPPlugin@1.0) that checks the correctness and completeness of SIPs, according to the E-ARK 2.0.4. specification. The service makes use of the Commons IP library.[2]

The results of the check are stored in a PREMIS event of type *well formedness check*. SIPs that fail the test are not accepted in the repository.

1.6 The repository shall obtain sufficient control over the Digital Objects to preserve them.

During ingestion, each digital object is given a UUID and checksums generated by different algorithms. The generated metadata is indexed by a SOLR engine. Since, the preservation strategy of the repository is to keep at all times the original file, legal provisions in order to obtain permissions to change files in case of normalization actions, are not necessary.

[2] https://github.com/keeps/commons-ip

At the moment, the repository keeps XML schemes that are used by XML files in the schemes directory of the AIP. (See AIP definition for more information)

1.7 The repository shall provide the producer/depositor with appropriate responses at agreed points during the ingest processes.

The ingest process is managed by an ingest process entity within the SCALA repository. Each process is assigned with a UUID and linked to the SIPs it is processing. Multiple SIPs can be processed during an ingest process

The repository manager and producer can at all times assess the progress of the ingest process. When the process ends an email notification is sent to the producer. The SCALA ingest process is developed in such a way that no human intervention is required during the process.

Errors are handled on the level of SIP. If one SIP fails, the process won't stop but will continue to ingest the next SIP in the batch. A report is generated for each SIP in the batch and linked to the ingest process entity for assessment by the producer.

Upon finishing the ingest procedure, an AIP is created in draft status, waiting for evaluation by the producer. After controlling the AIP, the AIP is taken out of draft status and included in the repository's catalog.

All processes of all producers are kept in one central register that can be consulted in case of audits.

1.8 The repository shall have up-to-date records of actions and administration processes that are relevant to content acquisition

See 1.7 for all information.

2. Ingest: Creation of the AIP

2.1 The repository shall have for each AIP or class of AIPs preserved by the repository an associated definition that is adequate for parsing the AIP and fit for long-term preservation needs.

AIPs are created and stored in accordance with E-ARK 4.0.2, of which the specification can be found at: https://dilcis.eu/. You should read this before going further in the SCALA specification.

AIDA maintains an interpretation manual for its AIPS at SCALA GitHub.

SCALA's AIP class for storing basic born digital archives is SCALA OTHER TYPE.

Content information

Content Data Object

The content data object is in all AIP classes stored in the representation's data object. In accordance with E-ARK, there is a representation folder for each representation of the content data object. The original representation is marked in the AIP.json and METS.xml.

In the basic type, the original folder structure as submitted in the SIP is kept. This original folder structure is an essential property. Within the RODA system, it also functions as Packaging Information.

Detailed information: SCALA GitHub.

Representation information

The SCALA repository currently only stores IPs of OTHER type. The current preservation plan only implies storing of representation information in the form of file format identification in order to support migrations on a bulk level. The following information is being kept in accordance with PREMIS v 3:

- file format (formatName, formatVersion)
- Pronom RegistryKey
- MIME Type RegistryKey

These metadata are kept in the representation's preservation metadata. Detailed information: SCALA GitHub.

There are no further characterisation tools in place that can produce more representation information for computer files. When we will eventually implement them, their output will be stored in the PREMIS element objectCharacteristicsExtension, or in the 'other' metadata section.

The repository currently **does not** store representation information for complex objects, like websites. As stated above in the OTHER AIP class, each file will be considered one intellectual entity.

Preservation Description Information (PDI)

Reference Information

Each AIP is referenced by an UUID. The value is stated in the METS.xml of the AIP, in the root element's @OBJID attribute, according to E-ARK.

Within the content information, each representation and each file within the representation is referenced by a UUID. In addition, each file has a localid based on the filename. The references can be found in the representation's preservation metadata, as well as in the fileSec of the representation METS.xml.

Detailed information: SCALA GitHub.

Within the RODA preservation system, each preservation event, agent, process and job is identified with an UUID. When they are included in the AIP, the UUID is used as reference.

Provenance Information

The AIP includes a trail of all essential preservation actions that can have an impact on the integrity, authenticity, findability and useability of the AIP and its content information.

By convention within AIDA, preservation actions that do not impact integrity, authenticity etc. like successful integrity checks after ingestion, user logs… do not necessarily have to be included in the AIP.

Provenance information is only stored starting from the moment of ingest. Provenance information that applies to pre-ingest should be stored by the content provider in the AIP's documentation folder

PREMIS metadata

The core of an AIP's provenance information is constituted by metadata in the form of PREMIS events. Each PREMIS event is stored in a separate file and is linked to a representation or to a specific file/specific files, using PREMIS' linkingObjectIdentifier field. The link is not reciprocal. Detailed information: SCALA GitHub.

OTHER metadata

Another form of provenance information is constituted by logs generated by tools that operate as a preservation agent. These logs are stored in the representation's OTHER metadata. The storing of this information in the representation's OTHER metadata is not dictated by E-ARK, nor is it an established best practice. It is a convention created by the RODA software. Currently, only Siegfried is a tool that produces such provenance information. Detailed information: SCALA GitHub.

AIP.json

The AIP's AIP.json contains the UUID of the SIP that was used as a source to create the AIP. This information is however also stored in the PREMIS-events related to the ingest procedure.

Additionally, the AIP.json also stores the UUID of the ingest job that was used to create the AIP. This ingest job ID is not stored in other AIP components.

Provenance information regarding AIP editions

With AIP editions, we mean changes in the original content information of the AIP. For both AIP classes, this constitutes:

1. Adding new original files
2. Deleting original files
3. Changing the AIP's original folder structure

It is necessary with regard to the AIP's authenticity that these kinds of actions are thoroughly logged in the PREMIS metadata. Logs of these actions are kept in PREMIS metadata on the representation and AIP level.

Context Information

Context information SHOULD be primarily provided in the AIP's Descriptive Information. AIDA's only standard requirement is to include a set of minimal descriptive metadata documenting the essential context and providing a reference to the AIP's external Descriptive information (see Descriptive Information).

Each AIDA partner can additionally add context information by creating a hierarchy using parent-child AIPs. A parent AIP is an AIP without Content Information, only containing metadata.

A parent-child AIP relation is implemented by a child AIP referencing a parent-AIP. This is done in the AIP.json metadata file and METS.xml file.

Risk: ISO 16363 and E-ARK demand reciprocal relations. A parent-AIP should be referencing child AIPs. This is currently not the case.

An important piece of context information is the original file path of each file, relative to the original carrier. A SCALA AIP only retains the file path relative to the representation's data folder. A producer that wishes to add the original file path relative to the original carrier can add this element in a unitid element of the EAD.xml.

Risk: original file path relative to the original carrier is currently not obligatory. This may lead to loss of authenticity.

Each AIP MUST have a reference to the archival description. This is done by the Descriptive Information in the AIP's scala.xml. (See Descriptive Information).

This reference SHOULD be reciprocal. The archival description should contain a reference to the AIP using its UUID.

Fixity Information

Each content file stored in the AIP's representations has a checksum. Three kinds of checksums are calculated: MD5, SHA-1 and SHA-256.

These checksums are stored in the object's PREMIS metadata. Detailed information: SCALA GitHub.

Checksums of metadata files will only be calculated when exporting the AIP when synchronizing with meemoo storage. From that moment, a checksum for the whole AIP will be calculated.

The checksums of metadata files will be stored (together with checksums of content files) in the METS.xml according to the E-ARK standard.

When synchronizing to meemoo, an md5 manifest for the whole AIP package will be generated. This is stored in meemoo's mdProperties field (value: `md5_viaa`), outside the AIP.

Access Rights Information

Access Rights Information is primarily stored outside the AIP-package, in the external Descriptive Information.

This mostly implies the AIDA-partners’ archive management systems. Each AIDA-partner can use its own conventions for storing Access Rights Information.

Within the AIP’s scala.xml file, storing the Descriptive Information, one field (‘accessrestrict’) is dedicated for storing basic Access Rights Information. However, this is not an obligatory field.

Within the AIP’s AIP.json file, access rights information is kept supporting the access of AIPs within RODA in the ‘permissions field.

```
"permissions": {

        "users": {

              "CREATE": ["ingest-OR-jq0st8z", "admin"],

              "READ": ["ingest-OR-jq0st8z", "admin"],

              "UPDATE": ["ingest-OR-jq0st8z", "admin"],

              "DELETE": ["ingest-OR-jq0st8z", "admin"],

              "GRANT": ["ingest-OR-jq0st8z", "admin"]

        },

        "groups": {

              "CREATE": ["Curator-VAi", "administrators"],

              "READ": ["Curator-VAi", "administrators", "VAi"],

              "UPDATE": ["Curator-VAi", "administrators"],

              "DELETE": ["Curator-VAi", "administrators"],

              "GRANT": ["administrators"]

        }

  }
```

This information will probably be essential in the case of an exit scenario.

A final place where access rights information is kept, is in meemoo’s MAM metadata. Accessibility is defined in the following boolean fields. The default settings are:

```
exportable: true,

editable: true,

deletable: false,
isPublic: false
```

Additionally, the AIPs can fall under meemoo’s licensing framework. These are stored in the mdProperties’ dc_rights_licenses field. The default settings when using meemoo as a CP are:

```
{
subKey: "multiselect",
value: [
{
value: "VIAA-ONDERWIJS",
attribute: "multiselect",
dottedKey: null},
{
value: "VIAA-ONDERZOEK",
attribute: "multiselect",
dottedKey: null},
{
value: "VIAA-INTRA_CP-CONTENT",
attribute: "multiselect",
dottedKey: null},
{
value: "VIAA-INTRA_CP-METADATA-ALL",
attribute: "multiselect",
dottedKey: null},
{
value: "VIAA-PUBLIEK-METADATA-LTD",
attribute: "multiselect",
dottedKey: null}
],
attribute: "dc_rights_licenses",
    dottedKey: null}
```

However, all settings in meemoo should be implementations of the original Access Rights Information stored in the partner's Archive Management Systems. In case of an exit-scenario, the meemoo MAM access metadata SHOULD NOT be essential.

AIDA will probably always opt for keeping access rights information outside the AIP-package, since this information is prone to change and since it is not essential for safeguarding the AIP's authenticity, integrity, useability and findability.

If however integration of Access Rights Information would be necessary within the AIP-package, this can be done by using PREMIS rights information or by adding a descriptive metadata file containing the access rights information.

Packaging Information

The packaging information is implemented according to the E-ARK standard, relying on a fixed folder structure and a METS.xml. The RODA implementation makes use of the divided METS structure (See divided METS structure in E-ARK CSIP), which is to be preferred when making AIPs with a lot of data.

It is in the METS.xml that a representation is denoted as an 'original' representation.

The AIP also contains an AIP.json that is actually used by the RODA system.

RODA makes use of the server's filesystem to store all the AIP's components. When exporting or synchronizing to meemoo, RODA packages the AIP in a ZIP format.

Within the AIP, the folder structure within a representation's data folder plays a role of major importance in constituting the logical package. The folder structure is kept as is and is documented in the fileSec of the representation METS.xml.

Currently, the folder structure within the data folder is replicated in the representation's technical metadata and other metadata folders and is used actively by the system for linking data and metadata.

When new representations of files are created, these will be kept in a new representation. The representation is created by adding a new directory within the *representations* directory.

Since a separation of different representations can only be made on the AIP-level and not on the file level, non migrated content will need to be copied to an alternative representation.

The current packaging conventions based on folder structure also pose risks regarding the handling of different representations.

- **Since a separation of different representations can only be made on the AIP-level and not on the file level, non migrated content will need to be copied to an alternative representation.**
- **It is not clear nor tested how the current packaging conventions will handle more than two different representations of an original file, since preservation PREMIS metadata are not registered on file level.**

Descriptive Information

Like Access Rights Information, Descriptive Information is kept primarily kept outside the AIP-package, in the AIDA partner's archive management systems. Each partner can create and store metadata according to its own conventions.

In accordance with ISO 16363, 4.5.2, we identified minimal Descriptive Information. This information is kept in a file scala.xml. Some fields, like the reference to the complete descriptive information are obligatory. Some of this information is also sent to meemoo when synchronizing the AIP. However, in case of an exit-scenario, the descriptive information kept in meemoo's MAM should not be essential.

Detailed information on SCALA GitHub.

AIDA will probably always opt for keeping Descriptive Information outside the AIP-package, since this information is prone to change and since it is not essential for safeguarding the AIP's authenticity, integrity, useability and findability.

If however integration of Access Rights Information would be necessary within the AIP, this can be done by using PREMIS rights information or by adding new descriptive metadata files containing the complete information. It is up to each partner to foresee a mapping from its conventions to the general AIDA convention. However, this general AIDA convention will only be established when necessary. Therefore, it is highly recommended that an AIDA partner uses established practices in the archival world, like EAD or RiC-CM.

Administrative information

Administrative information is not identified as a component by OAIS.

The AIP's administrative information is contained within a file called meemoo.xml, that is stored in the descriptive metadata.

The meemoo.xml contains all administrative metadata that support the current implementation with the meemoo storage. These metadata do not belong to the logical AIP, but are important for a correct management of the AIPs by RODA, for as long as we are using the meemoo storage service.

2.2 The repository shall have a description of how AIPs are constructed from SIPs.

AIP creation is done by transforming SIPs by RODA's default ingest process.

The data folders in the SIP's representations, the SIP's documentation folder and the descriptive metadata files are preserved in the AIP unchanged. During ingestion, Preservation Description Information is added, as well as descriptive metadata files. All the components are repackaged.

The ingest is finalized with an assessment by the archivist. His/her approval concludes the ingest process. In this stage, the AIP does not conform to E-ARK since the obligatory METS is absent. However, all information is present in the form of an AIP.json and the PREMIS.xml files.

Detailed information on SCALA GitHub.

After conclusion of the ingest process, the AIP is not automatically synchronized with the meemoo storage. A separate plugin should be run instead on the ingested AIPs. RODA's basic AIP will be updated with METS.xml information according to E-ARK.

After synchronizing to meemoo, an archivist can choose to PRUNE an AIP in order to liberate storage space. When pruning an AIP, the AIP's representations will be deleted. A pruned AIP can be restored by retrieving the AIPs representations from the meemoo storage.

All AIPS MUST be restored to RODA if preservation actions are carried out using RODA.

2.3 The repository shall document the final disposition of all SIPs.

After transforming a SIP to an AIP, the SIP is deleted.

If ingestion of a SIP fails, the original SIP is kept in RODA's transfer zone. An archivist can choose to retry the ingest process, or delete the SIP in RODA's transfer zone.

Administrative metadata about removal of SIPs are kept in the logs as a 'Delete Transferred Resource' action: f.e. https://scala.meemoo.be/#administration/log/logentry/b35b1736-bf5f-4f6b-9f1c-05d02d92c918

2.4 The repository shall have and use a convention that generates persistent, unique identifiers for all AIPs.

The repository uses UUIDs

All AIPs are identified by a UUID that is unique within the repository. Likewise, all AIP components have UUIDs that are referenced in the AIP's packaging information (METS.xml) See for more information 'Reference Information'

We do not foresee changes in identifiers at the moment.

No method for checking duplicate identifiers is in place. However, making use of an UUID reduces risks of duplicate identifiers to a bare minimum.

Since a UUID can be used to identify anything and are implemented worldwide, we believe the solution is adequately robust for the future.

The system uses the AIP's UUID as reference for all operations that affect AIPs. Also AIP components can be referenced, f.e. this example.

Since AIDA uses the OTHER AIP types to store large amounts of raw data, a chance exists that the content information gets changed in future, by deleting files from the AIP, or by restructuring AIPs due to a new archival description. This is very possible, regarding the relation between logical AIP and the archival unit of description.

In case of deletion or creation of data in an AIP, the UUID of the AIP will remain unchanged. However, if an AIP needs to be restructured, a new AIP will be created with a new UUID.

Risk: there is no method in place to trace the UUID of older AIPs.

2.5 The repository shall have access to necessary tools and resources to provide authoritative Representation Information for all of the digital objects it contains

The representation information is generated by Siegfried.[3]

Detailed information about Siegfried's configuration: Scala Github

2.6 The repository shall have documented processes for acquiring Preservation Description Information (PDI) for its associated Content Information and acquire PDI in accordance with the documented processes.

All processes in the SCALA ingest flow create the necessary PDI to guarantee authenticity, integrity, findability and useability of the AIPs.

The manner in which each preservation action in the SCALA ingest workflow is documented in PREMIS metadata is documented in the SCALA Github.

2. AIP samples

Basic AIP samples for reference are kept on Scala GitHub. It contains three samples

- absolute minimum AIP (an AIP only containing metadata)
- an AIP with an original representation
- an AIP with an original + normalized representation

[3] https://github.com/richardlehane/siegfried