D4.2 Tools to validate and upgrade assets

Grant Agreement nr	780470
Project acronym	SAUCE
Project start date (duration)	January 1st 2018 (36 months)
Document due:	June 30th 2019
Actual delivery date	June 28th 2019
Leader	DNEG
Reply to	William Greenly - wmg@dneg.com
Document status	Submission Version

Project funded by H2020 from the European Commission

Project ref. no.	780470
Project acronym	SAUCE
Project full title	**S**mart **A**sset re-**U**se in **C**reative **E**nvironments
Document name	D4.2 Tools to validate and upgrade assets
Security (distribution level)	Public
Contractual date of delivery	June 30th 2019
Actual date of delivery	June 28th 2019
Deliverable name	Tools to validate and upgrade assets
Type	Demonstration
Status & version	Submission Version
Number of pages	18
WP / Task responsible	DNEG
Other contributors	-
Author(s)	William Greenly - DNEG
EC Project Officer	Ms. Cristina Maier, Cristina.MAIER@ec.europa.eu
Abstract	Vocabulary and tools to describe deprecated assets and provide methods to upgrade and update them
Keywords	Validate, deprecate, upgrade, update, transformation, search
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
1.0 01-06-19	document created by William Greenly
1.1 23-06-19	Version for peer review
1.2 24-06-19	Revisions in response to review, final submission to EC

Table of Contents

1 EXECUTIVE SUMMARY

This document and deliverable provides a detailed overview of the different ways that we can validate and upgrade assets in order to maximise their potential for reuse.

We begin by carefully referencing the relationship between this deliverable and previous deliverables since the technology building blocks described in this document are dependent upon the search framework from work packages 4 and the transformation framework in work package 5. Without visibility of these documents and examples, it is not possible for this deliverable to be fully realised.

We then proceed to give an overview of the deprecation vocabulary and framework, along with providing detailed examples of different types of deprecation. This supplies the means by which different types of deprecation can be represented and classified, and as we shall discover, upgraded by the transformation and classification frameworks. We demonstrate using detailed code examples, different ways of representing and describing different types of deprecation using an expressive and interoperable description logic, entirely similar to that used in the search and transformation framework.

By providing a means to describe different types of deprecation, we proceed to explain how to leverage the transformation framework to take advantage of available transformations in order to upgrade and update assets. We provide examples of different tools and different methods by which asset managers can interact with the transformation and search framework in order to update and upgrade assets. We also present the classification framework as a means to keep assets up to date and preserve their potential for reuse. By presenting multiple options for upgrade, we allow different VFX houses to choose the method that best suits their processes and policies.

2 BACKGROUND

This work package relies heavily on D4.1 Smart Search Framework and D5.3 Basic Capability to enable Asset Transform. Both of these frameworks underpin and provide the means to validate and upgrade assets, so it is essential that the reader has a strong understanding of both these frameworks in order to assess this document.

Since the delivery date of D5.3 coincides precisely with the delivery date of this document, it is expected that the reviewer will have access to the final draft of D5.3, since this deliverable cross references capabilities in it. D4.1 has already been reviewed and submitted so no additional consideration is required.

3 INTRODUCTION

This chapter provides the main objectives of this deliverable along with the methodology we have adopted to deliver on the objectives.

3.1 Main objectives and goals

The main objectives of this deliverable are

- To provide a framework for describing and advertising assets that are deprecated and invalid so that they can be upgraded

- To provide the means for flagging different types of deprecation (data type, missing metadata, outdated etc.) so that appropriate action can be taken.
- To provide options and capabilities for automatically upgrading assets using the transformation framework

3.2 Methodology

To ensure that this deliverable meets the objectives, we have produced a number of examples that prove we can deprecate and upgrade a variety of assets with different characteristics. These examples are all demonstrable using the framework and tools outlined in WP4.1 and WP5.3 and this deliverable extends the former by providing the following:

- Several upgrade actions which demonstrate and prove the platform's upgrading and pipelining capabilities, along with integration with an abstract storage system. These are part of the current Github repositories and are demonstrated in section 5.1
- A simple command line interface which demonstrates how upgrades can be performed based on different types of deprecation, which is also part of the project Github repository and demonstrated in 5.2 and 5.3
- Comprehensive documentation covering the deprecation and upgrade vocabularies, detailed in section 4.

3.3 Convention

This document contains many snippets of code, which are pre-formatted accordingly to the most suitable code style with a black background. In some cases bold italics of code snippets has been done for the purpose of highlighting something important.

4 Validating Assets

In D4.1 we introduced the ingestion and search framework. This is an important and foundational part of this deliverable since in order to deprecate assets, we must first ingest them and validate them. During the ingestion and classification process, asset type specific classifiers and validators will introspect assets and extract important metadata that will then be used to determine whether an asset is deprecated. Media specific criteria, depiction information, type specific metadata and provenance information are all important criteria by which we will determine whether an asset is deprecated and needs to be upgraded, and the ingestion and classification framework provides a means by which this data can be supplied.

Furthermore, whilst ingestion and classification is a major dependency for determining and discovering whether an asset is deprecated, using vocabularies and ontologies explained in the next section, as we shall see, it is not the only way, and for more advanced comparisons between assets, which may be difficult to determine at ingestion and classification time, custom predicates can be used to identify assets which are no longer fit for purpose, reuse or composition.

5 Deprecating Assets

A deprecated asset is an asset that can no longer be used within an existing VFX pipeline, due to characteristics of the asset that are no longer compatible with current or evolving toolsets and tool chains. Some of these characteristics can include:

- The data type or file format is no longer supported, or has become incompatible with some other file format or software agent.
- The asset is missing metadata or important information that some other tool or process requires for reuse.
- The asset is explicitly defined as too old and out of date due to timestamp, provenance or revision related information.

In D5.3 we introduced a new vocabulary for describing asset representations and associated transformation actions. In this work package we will extend this vocabulary with terms to describe representations that have been deprecated, and in exactly the same theme as D5.3, we will show how we can register transformation actions to upgrade them. It is only by introducing rules to describe aspects that are deprecated, that the framework can decide what transformations are available for upgrade.

In D5.3 we introduced a new class which was the AvailableRepresentation, from which a number of subclasses were derived, each describing necessary and sufficient conditions for qualification. In the same spirit, we will create a new class of representations called DeprecatedRepresentation.

```
scet:DeprecatedRepresentation a owl:Class;
    rdfs:label "DeprecatedRepresentation";
    rdfs:comment "Representations that are deprecated".
```

In the succeeding sections, we will show how we can extend this class to describe different types of deprecation. It is important that we separate the different types of deprecation, since production houses may want to provide different upgrade paths and different upgrade mechanisms for different types of deprecation.

5.1 Deprecating by file format

Deprecating file formats is achieved in an almost identical way as the AvailableRepresentation examples for file formats that we used in Section 5 in WP5.3

```
scet:AvailableZMDLRepresentation a owl:Class;
       rdfs:label "AvailableZMDLRepresentation";
       rdfs:comment "A proprietary ZMDL representation";
       rdfs:subClassOf scet:AvailableRepresentation.

scet:ZMDLDistribution a owl:Class;
       rdfs:label "ZMDLDistribution";
       rdfs:comment "A ZMDL distribution";
       rdfs:subClassOf [
       a owl:Restriction;
```

```
      owl:onProperty dcat:mediaType;
      owl:hasValue "application/zmdl";
      rdfs:subClassOf scet:AvailableZMDLRepresentation
      ].

scet:DeprecatedFileFormatRepresentation a owl:Class;
      rdfs:label "DeprecatedFileFormatRepresentation";
      rdfs:comment "File formats that are deprecated";
      rdfs:subClassOf scet:DeprecatedRepresentation.

scet:AvailableZMDLRepresentation rdfs:subClassOf
scet:DeprecatedFileFormatRepresentation.
```

In the example above we use the zmdl file format, a DNEG proprietary form, to demonstrate deprecation. We do this by first registering the file format and available representation into the framework, followed by creating a class of deprecations for all file formats. Finally we define a specialisation of this class which contains all zmdl representations. It should be noted that the mediaType property can contain version number information, and the most comment convention is to use a + sign with the version of the filetype e.g "application/maya+2013".

5.2 Deprecating by media-specific criteria

In the section above, we provided an example of how to deprecate assets based on their file/data/media type, however for specific media types, there will be additional data about the asset which is important, to determine whether the asset is fit for reuse and transformation. Examples of these include the specific version of the media type, compression or resolution, serialisation format (ascii, binary) and software or hardware dependencies for rendering. In order to describe these, media-specific criteria, vocabularies with media specific namespaces can be created to capture media-specific metadata, much in the same way as classifiers create dynamic ontologies on ingestion.

```
@prefix maya: <https://www.autodesk.com/products/maya/>.
@prefix usd: <https://graphics.pixar.com/usd/>.

maya:mediaVersion a owl:DatatypeProperty;
      rdfs:domain scet:AvailableRepresentation;
      rdfs:range xsd:integer.

usd:mediaFormat a owl:DatatypeProperty;
      rdfs:domain scet:AvailableRepresentation;
      rdfs:range xsd:string.

scet:MediaSpecificDeprecatedRepresentation a owl:Class;
    rdfs:label scet:MediaSpecificDeprecatedRepresentation
    rdfs:comment "Representations that a deprecated due to
media-specific criteria";
```

```
    rdfs:subClassOf scet:DeprecatedRepresentation.

scet:MediaVersionDeprecatedRepresentation a owl:Class;
    rdfs:label "MediaVersionDeprecatedRepresentation";
    rdfs:comment "Representations that have the wrong media version";
    rdfs:subClassOf [
       rdfs:subClassOf scet:MediaSpecificDeprecatedRepresentation;
       a owl:Restriction;
      owl:onProperty maya:mediaVersion;
      owl:allValuesFrom [
            a rdfs:Datatype;
            owl:onDatatype xsd:integer;
            owl:withRestrictions ([xsd:maxInclusive 2010])
      ]
    ].

scet:MediaFormatDeprecatedRepresentation a owl:Class;
    rdfs:label "MediaFormatDeprecatedRepresentation";
    rdfs:comment "A media-specific format deprecation";
    rdfs:subClassOf  [
        a owl:Restriction;
        owl:onProperty usd:mediaFormat;
        owl:hasValue "usdb";
        rdfs:subClassOf scet:MediaSpecificDeprecatedRepresentation
    ].
```

In the example above we create prefixes for new media-specific criteria and create properties for both the maya version and usd format. We then create a new class of deprecation for media-specific criteria along with two specialisations, one which describes Maya representations which are considered deprecated due to their Maya version, and one which describes USD's which are deprecated due to their USD format, in this case the binary representation.

It should be clear that it is important to ensure correct namespaces and prefixes are provided, so as to distinguish between different criteria for different media types.

5.3 Deprecating by asset metadata

In many asset management systems, the lack of metadata about an asset or asset representation disqualifies it from being reusable or non transformable to a representation that is suitable. In addition the presence of certain metadata might also qualify the asset as deprecated due to some characteristic associated with the asset representation.

```
scet:MissingMetadataDeprecatedRepresentation a owl:Class;
    rdfs:label "MissingMetadataDeprecatedRepresentation";
    rdfs:comment "Representations that a deprecated due to missing metadata";
```

```
    rdfs:subClassOf scet:DeprecatedRepresentation.

scet:MissingColourChannelsDeprecatedRepresentation a owl:Class;
    rdfs:label "MissingColourChannelsDeprecatedRepresentation";
    rdfs:comment "Representations that lack colour channels";
    rdfs:subClassOf [
         a owl:Restriction;
        owl:onProperty scec:colour-channels;
        owl:maxQualifiedCardinality "0"^^xsd:nonNegativeInteger;
        owl:onClass scet:AvailableAlembicRepresentation
    ].

scet:MetadataCharacteristicDeprecatedRepresentation a owl:Class;
    rdfs:label "MetadataCharacteristicDeprecatedRepresentation";
    rdfs:comment "Representations that have features or characteristics that are
deprecated";
    rdfs:subClassOf scet:DeprecatedRepresentation.

scet:LowColourChannelDeprecatedRepresentation a owl:Class;
    rdfs:label "LowColourChannelDeprecatedRepresentation";
    rdfs:comment "Representations that have to few colour channels";
    rdfs:subClassOf [
        scet:MetadataCharacteristicDeprecatedRepresentation;
        a owl:Restriction;
       owl:onProperty scec:colour-channels;
       owl:allValuesFrom [
             a rdfs:Datatype;
             owl:onDatatype xsd:decimal;
             owl:withRestrictions ([xsd:maxInclusive 256])
       ]
    ].
```

In the example above we declare a class of deprecations that are representations without certain metadata, and then provide a specialisation of this which contains things that don't have colour channel metadata.

In the second example we show how we can use metadata to identify deprecated representations, in this case representations that have too few colour channels.

5.4 Deprecating by depiction

In some circumstances the asset may become deprecated due to the nature of the contents of the things it is depicting. In Section 5.3.7 of D4.1 we explained how asset depictions can be modelled and described using a variety of upper ontologies, in this case YAGO (YAGO incorporates Google, Wordnet and WikiData). In many production houses, fundamental aspects of these depictions render them deprecated and no longer reusable, especially in the case when they need to be combined with other assets:

```
scet:DepictionDeprecatedRepresentation a owl:Class;
      rdfs:label "DepictionDeprecatedRepresentation";
      rdfs:comment "Deprecations that are as a result of changes to
depictions".

scet:VertebralColumnDeprecatedRepresentation a owl:Class;
      rdfs:label "VertebralColumnDeprecatedRepresentation";
      rdfs:comment "Deprecation due to spines with 12 bones";
      rdfs:subClassOf[
      a owl:class;
      owl:complementOf [
      a owl:restriction;
      rdfs:subClassOf scet:DepictionDeprecatedRepresentation;
      owl:onProperty dc:about;
      owl:minCardinality 1;
      owl:onClass [
      owl:intersectionOf (
            yago:Human_vertebral_column
            [
                  a owl:restriction;
                  owl:onProperty dct:hasPart;
                  owl:qualifiedCardinality 12;
                  owl:onClass yago:Bone
            ]
            )
      ]
      ]
      ].
```

In the example above we create a superclass of deprecations called DepictionDeprecatedRepresentation which contains all representations that are deprecated due to their depictions. We then create a specialisation of this class which contains all representations deprecated due to depictions of human vertebral columns (spines) not having exactly 12 bones.

In some cases the nature of the depiction is too complicated to classify and in such circumstances, materlisations should be achieved using custom predicates, alluded to in section 4.6 and detailed in D4.1, section 6.6.

5.5 Deprecating by revision, version or other provenance related information

Deprecating by revision, version or provenance works in exactly the same way as the previous examples, save that in this instance the provenance data is used to provide the necessary and sufficient conditions for deprecation.

```
scet:DeprecatedProvenanceRepresentation a owl:Class;
      rdfs:label "DeprecatedVersionRepresentation";
      rdfs:comment "Versions that are deprecated";
      rdfs:subClassOf scet:DeprecatedRepresentation.

scet:DeprecatedDateRepresentation a owl:Class;
      rdfs:label "DeprecatedDateRepresentation";
      rdfs:comment "Deprecated date representations";
      rdfs:subClassOf [
      a owl:Restriction;
      rdfs:subClassOf scet:DeprecatedProvenanceRepresentation;
      owl:onProperty dct:created;
      owl:allValuesFrom [
            a rdfs:Datatype;
            owl:onDatatype xsd:dateTime;
            owl:withRestrictions ([xsd:minInclusive
"2010-01-01T00:00:00"^^xsd:dateTime])
      ]

      ].
```

In the example above, we create a class of deprecations call DeprecatedProvenanceRepresentation and then a specialisation of this class called DeprecatedDateRepresentation which includes as its members all the representations that were last updated before a specific datetime. In this scenario we could also use OWL datatype properties to express the datetime as a range or a value from the current time.

5.6 How deprecations are materialised

The examples above are modelled using Web Ontology Language, and in order to materialise these deprecations, we must apply a reasoner. A reasoner applies the constraints and rules to the data and materialises all the logical conclusions and deductions. The complete set of rules and constraints is defined below:

https://www.w3.org/TR/owl2-primer/

There are a number of different ways the framework applies reasoning in a number of different contexts. Firstly, the graphstore and SPARQL processor applies backward chaining reasoning at query time to materialise data and we can use SPARQL property path expressions to traverse class relationships. In addition to this, the ingestion and classification framework also provides reasoning via the general purpose OWL reasoner, which is consumed as an action at ingestion and classification, and provides forward chaining materialisations. The combination of forward chaining for performance and backward chaining for flexibility, allows us to pick the materialisation strategy that best suits the specific vocabulary, its complexity and its likelihood to change.

Backward chaining happens automatically and is part of the graphstore and query processor. We can apply forward chaining periodically using the OpenWhisk scheduler or some other trigger mechanism (e.g the application of a new deprecation vocabulary). Both mechanisms are already incorporated into the OpenWhisk, the underlying platform.

Finally in addition to materialisations provided by the reasoner, we can also use custom predicates at query time to perform more complex, non-linear algorithms, delegated to various subsystems, to determine whether assets qualify as deprecated.

All the above methods of materialisation are described in more detail in section 6.6 in D4.1

6 Upgrading Assets

In the previous chapter we discussed how assets can be flagged or advertised as being deprecated, based on varying criteria across a range of asset characteristics. In this chapter we will describe the different ways by which the transformation framework detailed in D5.3, can be leveraged to upgrade ands and ensure their value is retained and maximised. It is therefore essential for the reader to have a comprehensive understanding of this framework since it provides the basis for upgrades.

It's also worth noting that whilst this document provides a number of different means by which users and asset management systems can interact with the framework to upgrade assets, these are by no means exclusive, and since the framework provides a rich and comprehensive API, any number of integrations and interactions can be made based on specific requirements and use cases.

However the ones listed below allow us to form an understanding of some of the mechanisms available by which we leverage the transformation framework to upgrade and update assets.

6.1 Creating an upgrade transformation

Upgrades are performed in exactly the same way as any other transformation in the framework, by creating transformation actions, and publishing and registering them in the framework. In the example below we will create a transformation action that can upgrade an asset with ZMDL representations, to a more modern file format such as Alembic. We will do this by creating a Docker based transformation action and registering its capabilities and relationship to deprecated file types, into the framework.

As described in section 6 in D5.3, a Docker transformation action consists of 2 main parts; the first part is the executable which is run by the Openwhisk Framework, the second is the Docker image in which the executable runs.

The executable below takes an asset input, downloads the deprecated representation, transforms it, saves the results and updates the search to ensure that it is no longer deprecated.

```
#!/bin/bash
```

```
# Extract parameters
echo "$1" > "params.json"
uuid=`jq '.scet:uuid' params.json | sed -e 's/^"//' -e 's/"$//'`
asset=`jq '.scet:asset' params.json | sed -e 's/^"//' -e 's/"$//'`
distribution=`jq '.scet:distribution' params.json | sed -e 's/^"//' -e
's/"$//'`
transformation=`jq '.scet:transformation' params.json | sed -e 's/^"//'
-e 's/"$//'`

# Download Data
mkdir /data
wget -O /data/input.zmdl "${distribution}?token=f9403fc5f537b4ab332d"

# run upgrade
zmdl2alembic /data/input.zmdl --out /data/output.abc
sleep 4

#upload data
hst=`host storage`; if [ ${#hst} -gt 0 ] ; then store="storage"; else
store="localhost"; fi
upload=`curl -X PUT -Ffile=@/data/output.usda
"http://${store}:25478/files/${transformation}-${uuid}.abc?token=f9403fc
5f537b4ab332d"`

#update search
./update.groovy $1

#return result
if [ -f "/data/output.abc" ]; then echo "{ \"result\": \"upgraded\" }";
else echo "{ \"result\": \"error\" }"; fi
```

In the case of the transformation action above, we discard the old representation, but it is entirely up to the action implementation and storage system to decide what should be done with the deprecated representation.

Now that we have created our action, we can build, test and publish in exactly the same way as described in section 6 in D5.3.

To deploy the action we can run the following

```
wsk -i action update zmdl2alembic exec.zip --docker
sauceproject/sauce-action-transformation-zmdl2alembic
```

We can now execute the upgrade using the transformation framework as follows:

```
wsk -i action invoke zmdl2alembic -P params.json
```

This will upgrade the asset and store the upgraded representation in the storage system

Upgrade pipelines

In some situations, multiple transformations are required to upgrade an asset, with intermediary representations that are also deprecated. In the same vein as transformation pipelines describe in D5.3, upgrade pipelines can provide the means by which multiple, sequenced transformations can be applied to upgrade an asset to an up to date representation.

In the example above, we created an upgrade transformation that converted deprecated ZMDL files to Alembic files. In some cases Alembic files are also deprecated in favour of USD representations, so a further step is necessary, which involves transforming from Alembic to USD.

In D5.3 we showed how this is possible using transformation actions and in exactly the same way as described in D5.3, we can sequence 2 actions together to create a transformation pipeline

```
wsk -i action update upgradeZmdl --sequence zmdl2alembic, alembic2usd
```

Implicit vs explicit upgrades

In the example above we explicitly declared an upgrade pipeline using a sequence of transformation actions, however in exactly the same way as transformation pipelines can be inferred by chaining actions together using the reasoner, so upgrade pipelines can be deduced in exactly the same way. In some circumstances this may be preferable or mandatory since the steps necessary to upgrade can only be discovered by non trivial means and are not obvious.

6.2 Discovering deprecated assets

In section 4 we described some different ways to describe classes of assets that were deprecated due to a variety of characteristics. In order to upgrade them, there needs to be a way that these deprecated assets can be discovered. Since information about deprecation is integrated with the search framework, we can use the search API (defined in D4.1, section 6) to expose assets that are deprecated:

```
SELECT distinct ?assetId ?representation WHERE {
      ?assetId a sce:VFXAsset.
      ?assetId dcat:distribution ?representation.
```

```
      ?representation a scet:DeprecatedRepresentation.
}
```

The query above lists assets that have deprecated representations. We can get specialisations of these deprecations by specifying the sub-type of deprecation in the query:

```
?representation a scet:MissingMetadataDeprecatedRepresentation.
?representation a scet:DeprecatedProvenanceRepresentation.
?representation a scet:DeprecatedFileFormatRepresentation.
?representation a scet:PipelineXDeprecatedRepresentation.
?representation a scet:MetadataCharacteristicDeprecatedRepresentation.
```

In the examples above, we list assets by the more specialised types of deprecation, defined in section 4.

We have also provided a CLI which wraps the search service and allows for more simplified interrogation:

```
./sauce list deprecated [deprecation-type]
```

This is explained further later in the chapter

Discovering deprecation using custom predicates

It is also possible to discover deprecated assets without using the reasoner and vocabularies, by using custom predicates. Custom predicates were mentioned in D4.1, section 6.3.7 for advanced search purposes for similarity. We can also use these for things like incompatibility, in order to work out what is deprecated automatically at query time, without relying on metadata or the reasoner / rules engine.

```
SELECT ?x
WHERE {
  ?x softwarenamespace:isNotCompatibleWith ?y.
  ?y a scet:6BoneSpine.
}
```

In the example above the *'softwarenamespace:isNoLongerCompatibleWith'* is a custom predicate that is registered with the search that delegates to a sub-system (i.e the software) which opens up resource ?x and compares to resource ?y, which in this case is a 6 bone spine.

6.3 Executing upgrades

Now that we can create and register upgrades and upgrade pipelines, we will now explore some of the different ways that asset management systems and asset users can trigger

upgrades. As mentioned previously, the methods below are a few common suggestions, but more sophisticated and convoluted integrations can be made using the API.

Upgrading using the CLI

We demonstrated in D4.1 and D5.3 a simple CLI for interacting with the search and transformation framework, providing simple commands to find and list assets as well as provide transformation options for an asset. By extending this we can also provide a way to find deprecated assets and then present options for upgrade.

```
./sauce list deprecated
```

This provides a list of assets that have representations that are deprecated. From the asset ID we can get the list of upgrade options for the asset as follows:

```
./sauce upgrade [asset-id]
```

As described in D5.3, actions names registered in the search framework should correspond to the action names created in OpenWhisk. We can now upgrade the asset by running the upgrade transformation:

```
wsk -i action invoke [action] -P params.json
```

As mentioned in the previous section, this can also be applied to specific types of deprecation. Below is an example for assets that are outdated as described in section 4.3

```
./sauce list deprecated scet:DeprecatedDateRepresentation
```

We can use the CLI to show upgrade options in exactly the same way as demonstrated earlier.

Upgrading via ingestion / classification

As described in section 4, one of the main causes of asset deprecation, is the lack of asset metadata. The framework, along with many asset transformation tools, require metadata to determine qualification for transformation, and without the data the asset's potential for reuse is diminished. D4.1 provided examples of ingestion and classification pipelines, and provided a way that contributors can provide plugins for different types of assets and asset data in order to introspect and extract metadata. By using the CLI we can list assets that are missing metadata as follows:

```
./sauce list deprecated scet:MissingMetadataDeprecatedRepresentation
```

This lists all the assets that are classed as having missing metadata representations. We can reclassify this asset by re-ingesting it using the correct ingestion pipeline for the type of asset.

Upgrading using a scheduler

The OpenWhisk framework also provides a scheduler that can trigger transformations and actions. Since both classification and transformation are actions registered with the OpenWhisk framework, then periodic checks for deprecations resulting in automatic upgrades can be scheduled to run. In addition to the scheduler, OpenWhisk also has support for triggers, which can be activated by a plethora of internal and external events, thus proving additional options and opportunities for automatic asset upgrades.

Upgrading using a graphical user interface

Although not explicitly part of this deliverable or within the scope of D4.1 or D5.3, a more visual way to help asset maintainers understand the state of their assets, would be to provide a graphical user interface that allows users to see which assets are deprecated and then allow them to set policies and execute upgrade pipelines based on individual circumstances. This could be part of the storage system or asset management system and could simply leverage the same API's as the CLI.

7 Conclusion

Hopefully it should be apparent by this deliverable, that the transformation framework provides a flexible and extensible range of options to enable asset upgrades and updates, thus preserving and protecting the value of our assets for future reuse. By carefully defining different deprecation classes, we can configure the most suitable means by which to automatically upgrade and update assets based on available transformation capabilities. Furthermore we can also leverage the classification and ingestion framework to ensure we have important and valuable metadata about our assets that could otherwise prohibit them from transformation or reuse in a different context.

8 Web references

RDF
https://www.w3.org/standards/techs/rdf

RDFS
https://www.w3.org/TR/2014/REC-rdf-schema-20140225/

OWL2
https://www.w3.org/TR/owl2-overview/

SPARQL1.1

https://www.w3.org/TR/sparql11-query/

OpenWhisk
https://openwhisk.apache.org/

USD Toolset
https://graphics.pixar.com/usd/docs/USD-Toolset.html

Alembic
https://www.alembic.io/