Preservica

Logical Data Model

v6.9.0

Table of Contents

References

Document	Ref	Date	Details & Issue
Preservica Guide to System Documentation	[DOC]	Sept 2022	git/doc/UG/SUG V6.6.0
Reference Model for an Open Archival Information System (OAIS)	[OAIS]	June 2012	http://public.ccsds.org/publications/archive/650x0m2.pdf

Chapter 1. Introduction

1.1. Purpose of This Document

This document describes the logical entity data model used by Preservica.

1.2. Scope of This Documentation

This document describes the structure of the Logical Data Model used to represent entities, metadata and associated information by Preservica. It covers the design and high level structure of the data model.

1.3. Context of this Issue

This document is consistent with Preservica 6.9. Note that there is a major change in the entity data model in this version.

1.4. Definition of Terms

Term	Definition
DIP	Dissemination Information Package.
SIP	Submission Information Package.
XIP	XML schema used by Preservica to represent entities and metadata for SIPs and DIPs
OAIS	Reference Model for an Open Archival Information System.
PUID	Persistent Unique IDentifier (allocated in PRONOM).

Chapter 2. The Logical Entity Data Model

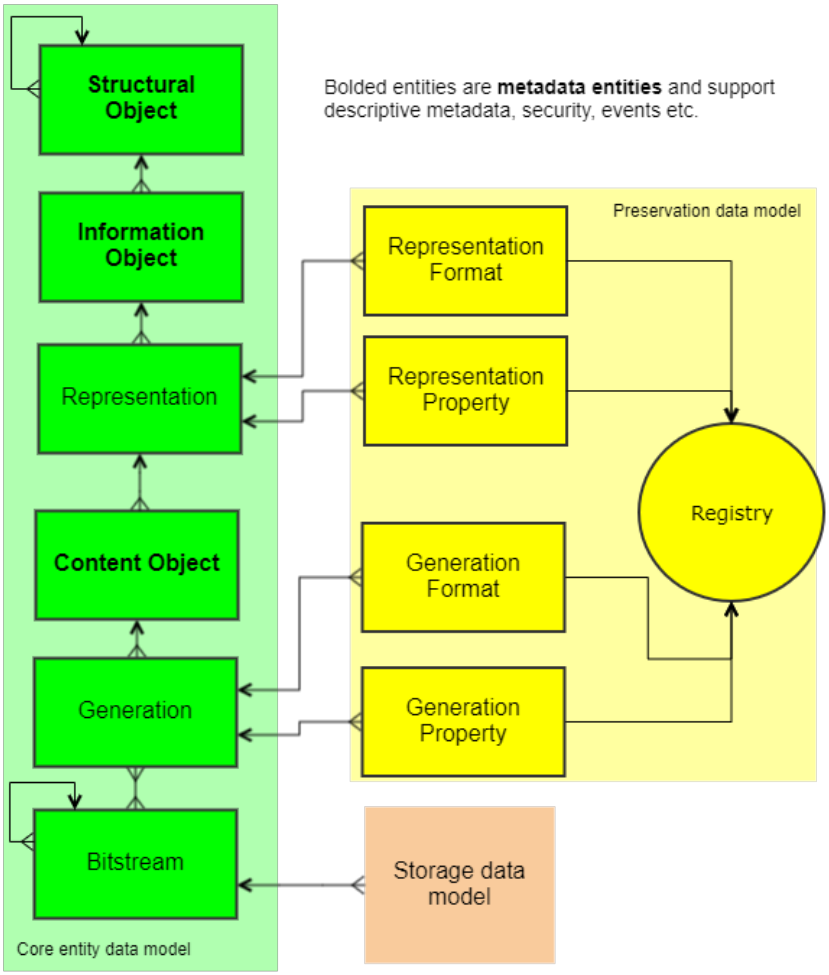
2.1. Core Model

Preservica's logical model is designed to be compatible with OAIS and PREMIS, while supporting flexible digital preservation and the presentation of the same information for different purposes. The OAIS model introduces a number of concepts that are expanded in the Preservica data model. In particular OAIS considers conceptual **Information Objects** that consist of **Content Information**, **Preservation Description Information**, **Packing Information** and **Descriptive Information**. There can be many types of Information Objects including different levels of aggregation. PREMIS described **Intellectual Entities** which can have multiple **Representations**, each of which contains content in the form of files.

In Preservica we expand this model to introduce a division between structure, i.e. where information should be placed within an archive, and information objects. We also add preservation information to allow information or content to be migrated, for obsolescence or presentation reasons.

This leads to a core model with six levels:

Figure 2.1. The core entity data model



The top two entity types are those which are most commonly seen by users:

- **Structural Object** (SO) or **Folder**: Structural objects make up the hierarchy within your archive. They can contain other structural objects, building up a tree structure, and also information objects.

- **Information Object** (IO) or **Asset**: A logically atomic piece of information, for example a picture or an email. Assets are the entities which you will generally interact with, for example by rendering or downloading them. An asset can't contain other assets – although it has substructure (see below), it is self contained and not hierarchical.

Beneath the information object are some further layers representing the detail of the asset.

- **Representation**: A way of viewing the information within an asset. The most common example of different representations would be a preservation copy and an access copy, for example a video in its initial format for preservation purposes, and a downscaled web-ready MP4 for access purposes. Information objects will always have at least one representation (the preservation representation).
- **Content Object** (CO): A logically atomic piece of content, for example an attachment on an email.
- **Generation**: Content objects contain generations, as a sequence of dated views of the content in different formats. When content is preserved, a new generation will be created, with the content in a less at-risk format. Only the most recent generation of content will be used by default (e.g. for retrieving content to render).
- **Bitstream**: The physical content, as stored on a storage adapter. In almost all cases, a generation of a CO will have only one bitstream; exceptions would include multi-part container files, or data formats where the data header and content are in separate physical files (which is rare).

The three metadata entities (SO, IO, CO) all support the following standard XIP properties:

- Ref
- Title
- Description
- Parent (a link to another entity)
- Security descriptor. In 6.0, this is a single security tag. For a CO, this should be the same as its parent IO.
- Custom type. On content objects, this field is set during characterisation to be the content type (e.g. Image).

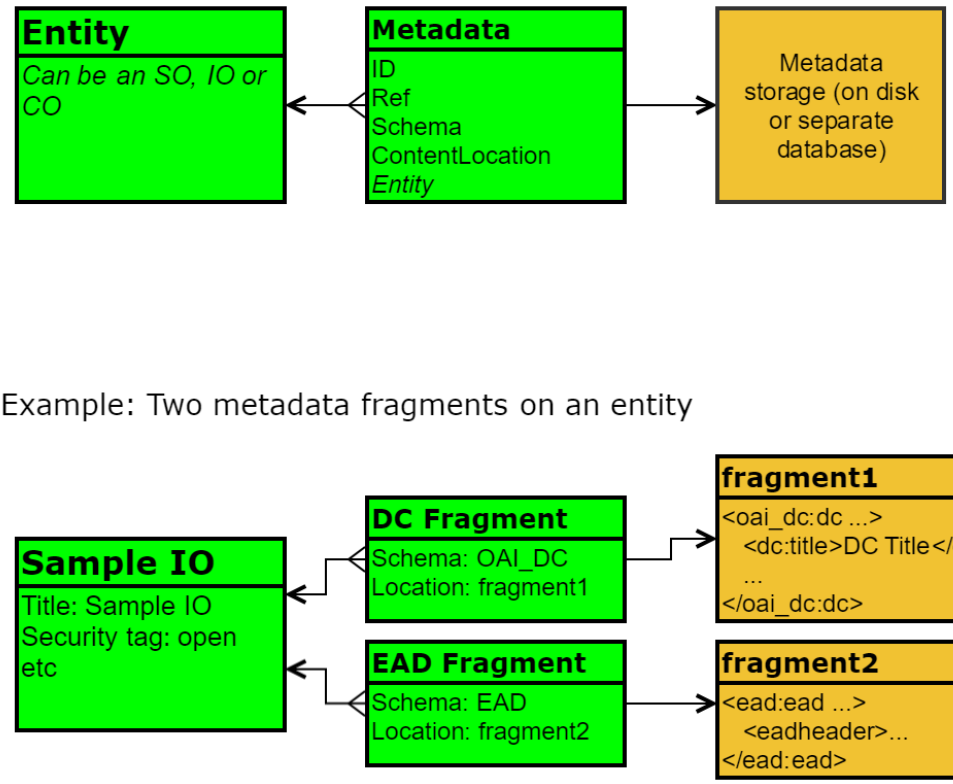
In many cases, an asset will contain one content object per representation. For example, ingesting a directory containing 5 spreadsheet files will result in 5 assets, one for each spreadsheet; each will contain a preservation representation with one content object. At ingest time, that content object will contain one generation, and the bitstream referenced from that generation will be the appropriate spreadsheet file.

An example where this is not the case is an email, when ingested from an email archive, and where the email has an attachment. In this case, the email body and its attachment will be two separate content objects, but as part of a single information object, and the email (as a single asset) will be shown as one item in Explorer.

2.2. Metadata

The three main entity types (structural, information and content objects) can have descriptive metadata fragments associated with them. Each fragment has a record in the core entity database, referring to the entity to which the fragment is attached, recording the schema URI and where the fragment is stored in the metadata storage system. The content of the fragment is stored outside the main entity database (either in another database schema, or on the file system, depending on your configuration).

Figure 2.2. How descriptive metadata is associated with entities



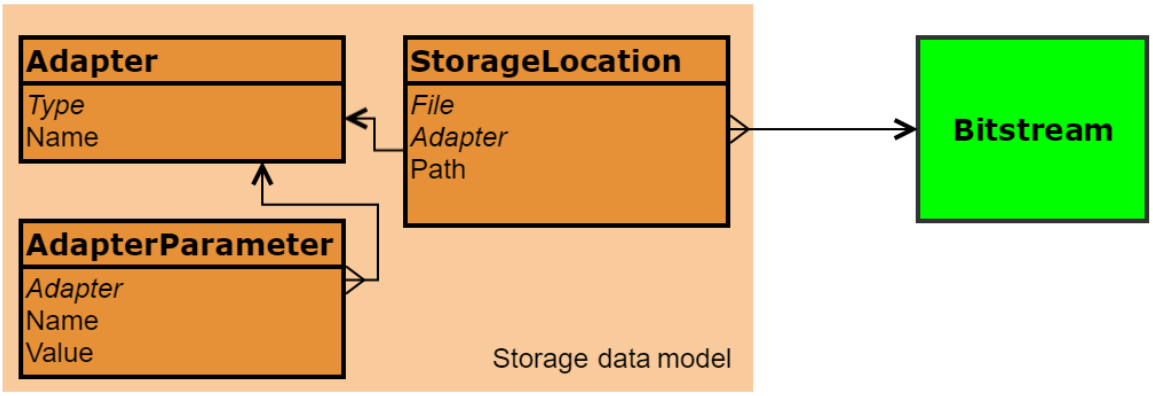
This model allows Preservica to determine which entities have metadata of a given schema, or which fragments an entity has, without having to read the content.

2.3. Storage Information

Only one part of this core model interacts with storage: the bitstream, which represents the actual file that is being stored and preserved. Preservica supports storage on multiple storage adapters, and *storage locations* define what path a bitstream is stored at on a storage adapter.

A bitstream may have a "container", which is another bitstream. This covers a case where a logical piece of content exists, but is currently embedded within another piece of content. The attachments to an MSG email message are one example of this. If a bitstream has a container, storage is handled by storage of the container and that bitstream is not separately persisted to any storage adapter.

Figure 2.3. The storage model



Adapter parameters configure a storage adapter to allow it to connect to its provider, and vary depending on the type of adapter.

2.4. Preservation Information

In this model, it is the **content object** – not the information object – which is aware of information about formats and preservation information. More precisely, each generation of a content object records which formats (as format PUIDs) that generation has been characterised with, along with the broader type of the main format. The content object itself records which type of content is present, more generically. A generation also records the technical metadata (characterisation properties) measured by any available characterisation tools.

For example, imagine ingesting a JPEG image which conforms to the JPEG standard version 1.02. The resulting asset will have one content object in its preservation representation, and that CO will have one generation. That generation will be characterised with a primary format of `fmt/43` (JPEG v1.02), and also – but at a lower priority – as `fmt/41` (RawJPEG Stream), a less specific format that it also conforms to. It will also record its format group as JPEG. The generation will also record properties extracted by tools like Jhove, for example image dimensions. The content object will record that it contains an image.

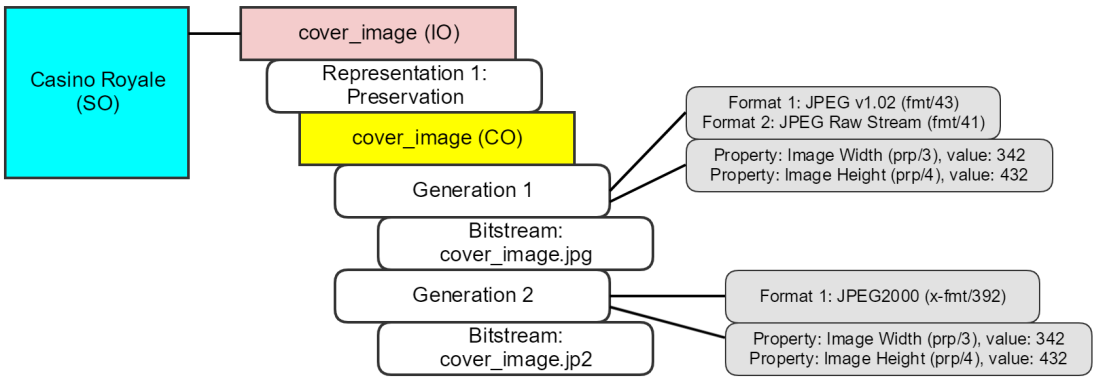
Information objects and representations do not hold any information about file formats; they are containers of logical information, and the technical information is stored with the content, not the information (i,e. on the content object and generations thereof, not the asset). However, representations can be aware of *representation formats*, defined by arrangements of content (for example an email with attachments), and properties associated with those formats.

The formats and properties, and information about the tools which should be used to identify and characterise files, come from the Registry through its PAR interface. Some of this information (format PUID, name and version; property PUID, group and name) are denormalised into the entity model, so it can stand apart from the Registry as a self-consistent model.

2.5. Migration

Preservica supports two types of migration: *preservation* (or *normalisation* in the PREMIS terminology), and the *creation of new representations*. Preservation creates new generations of content objects, with new bitstreams and new characterisation information. Creating a new representation will create new content objects in the new representation, referencing new bitstreams if a migration was necessary.

2.5.1. Example: Content Preservation

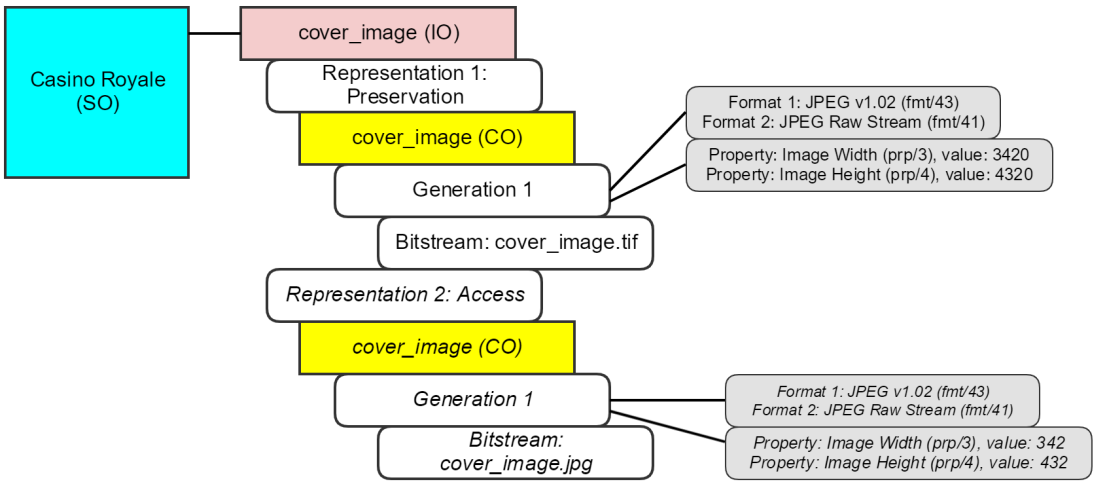


In this example a user has ingested a single information object, the cover image from the book *Casino Royale* as a JPEG image. Initially, that information object contained a Preservation representation, which had one content object with one generation (Generation 1). That generation had a bitstream, which is stored somewhere (see Section 2.3), and it was characterised with format information stating that it is a JPEG and with image properties.

Later, they have run a normalisation migration to preserve the JPEG, by migrating it to JPEG2000. This process creates a new generation (Generation 2), which has a new bitstream, and has been characterised with format information indicating that it is a JPEG2000 image.

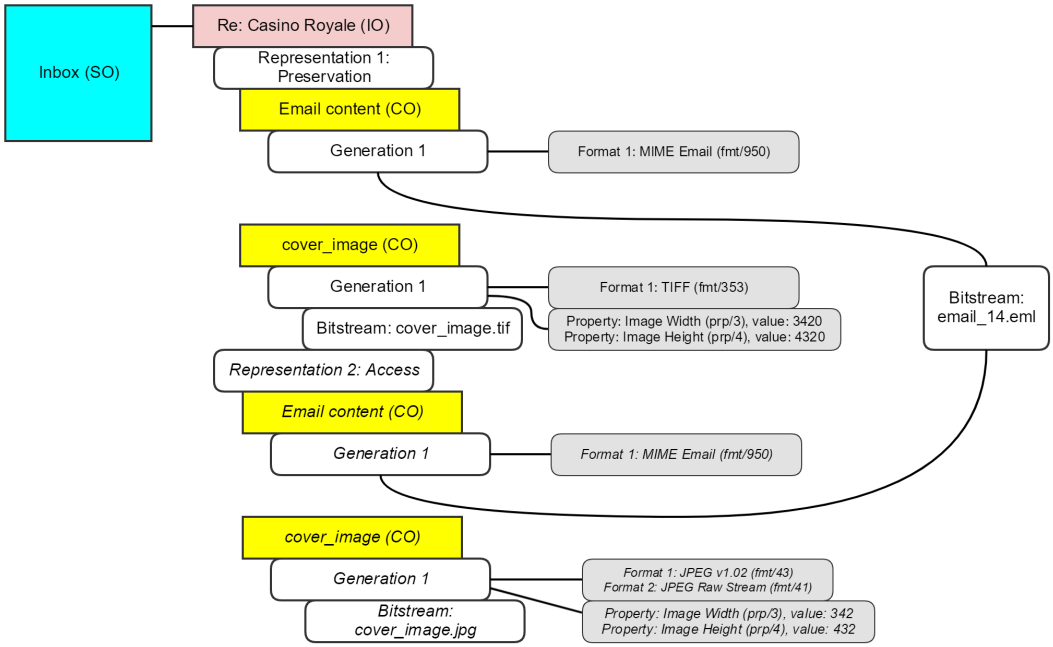
An alternative interpretation is that this asset was ingested with both generations defined in the ingest package. The model does not represent process information (although actions are recorded: see Section 2.7.1) or how the entities that are present got there.

2.5.2. Example: Multiple Representations



In this case the user has created two representations of the same content. The preservation copy is a high resolution TIFF, stored in the preservation representation. The access representation contains a low resolution JPEG suitable for displaying online. Each representation contains a single content object, with one generation, referencing the bitstream for the content file and recording the characterisation information.

2.5.3. Example: Bitstream Shared Across Representations



Here, the user has ingested an email with an attachment, creating an asset with multiple content objects in its preservation representation. The attachment is a high resolution TIFF image.

There is also a second representation, for example created using the manual workflow or with migration settings applying to ingests. In this access representation, the image has been migrated to a low resultion JPEG suitable for displaying online, as above. However, the email content (the .eml file) has not been migrated.

The bitstream for the email is shared between the two representations: the first generation of the relevant CO in each representation references it. The file is only recorded once, and stored according to the storage settings that apply to each representation (in the simple case, it is stored once).

2.5.4. Generation Order and Effective Dates

Normally, as in the content preservation example above, generations will be created in a logical order: later generations will be the result of a migration, which also makes them effective from that date, i.e. they will be considered to be "latest active" for the purposes of showing the current state of the content.

However, other processes may revise earlier generations (for example, in Preservica, recharacterisation can do this), and create new generations which refer to earlier content. This generation won't have an effective date, because the latest active generation should still be that created from migration.

2.6. Associated Information

As well as the core entities, there are several associated objects which reference an entity and add more information about it to the model.

- **Event Actions** record that something was done to an entity, and are grouped together into **events**; see Section 2.7.1.
- **Links** represent a non-hierarchical relationship between two entities. A link can be of any type, including user defined types. A link with the special type *VirtualChild* (from an SO to a target entity) will also cause Explorer to show the linked item in the Browse view of the link parent. Links don't have any security information or metadata associated with them.

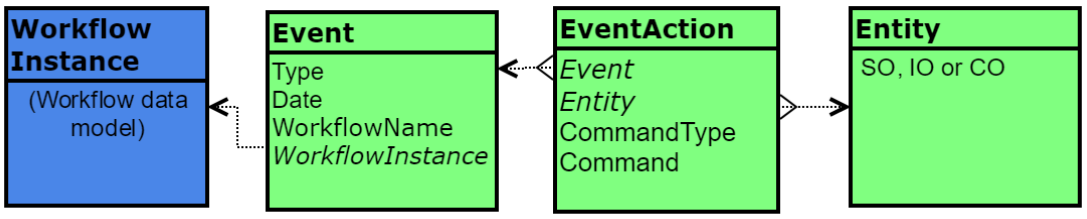
- **Identifiers** record how an entity is referenced in an external system, or through another mechanism, e.g. an ISBN number, or a URL, or an ID in a catalogue system. The type and value of an identifier are free text, but Preservica will automatically set some identifiers, particularly during catalogue synchronisation.
- **Restrictions** record restrictions that have been placed against entities. Currently, the only restriction type recognised by Preservica is `LOCK`, which represents an entity being locked for structural changes.
- **Previews** store the thumbnails and previews for an entity.
- **Deletions** record that an entity has been deleted (either fully or content only), before the deletion retention period has expired and final system deletion has been run. Deletion are grouped together into **deletion groups**, recording which entities were deleted together, and groups have **deletion relations** between them to ensure that restoration of deleted items leaves the system in a consistent state.
- **Fixities** are associated with a bitstream and record the fixity (checksum) values of the file content. Preservica supports multiple fixities, e.g. an MD5 and a SHA-256, being recorded against the same bitstream.

2.7. Audit Information

2.7.1. Events

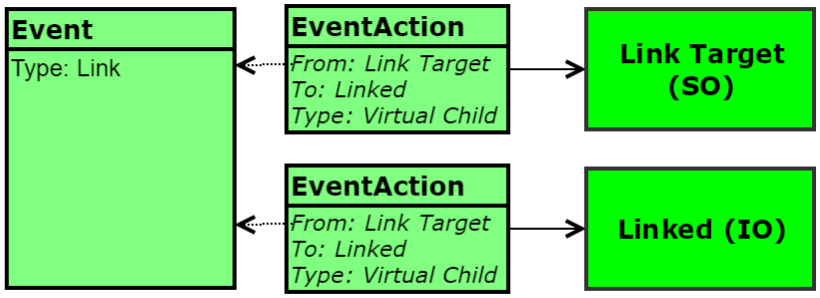
Most actions which take place against entities, either altering their state or retrieving information, are recorded in the event model.

Figure 2.4. How events and event actions are associated with entities



One event represents a logical action by the user, often associated with running a workflow. For example, adding a new fixity value to bitstreams via the Add Fixity workflow will create a single Add Fixity event. Linked to that event will be *event actions* for each entity that was affected by the event: in this case there will be an event action for each content object recording which fixity value was added to bitstreams inside that CO, and also an event action for the information objects affected. Most events will create an action on the asset, even if the actual change was made at a lower level, because only event actions recorded against the asset will show up in the Explorer history page.

Figure 2.5. Event with actions from creating a link



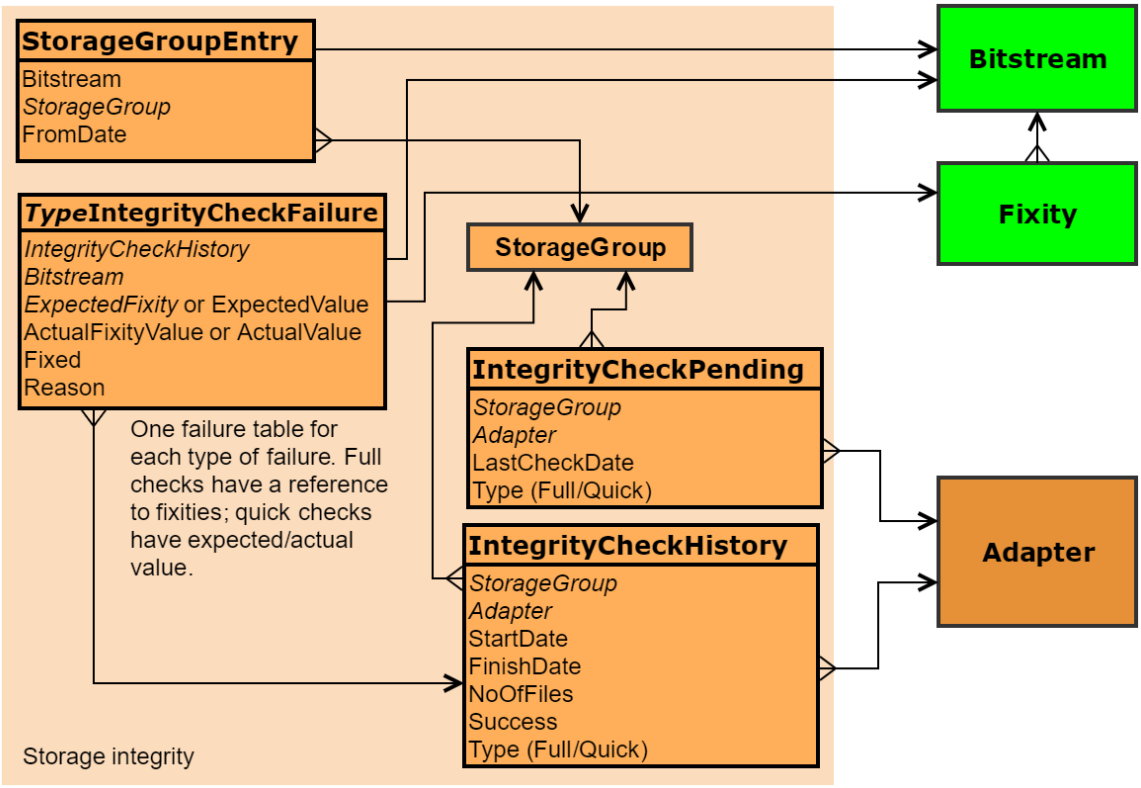
In this example we have created a virtual child link (see Section 2.6) from a link target folder to a linked asset. This creates one event, with two event actions, one for each of the two entities affected by the link. In this case the information stored on the actions is the same, but in general, it can be different.

In the *Command* field, event actions store information about what they did to the entity. For actions which can be rolled back, this follows the command pattern, i.e. there will be enough information stored to reverse the change. For example, the event action recorded when deleting a metadata fragment contains a copy of the fragment, so it can be restored.

2.7.2. Integrity Checking (Storage)

Integrity checks record when a file (bitstream) was checked for consistency. Preservica periodically checks each bitstream is correctly stored on each storage adapter it's supposed to be on, both *quick checks* (checking the file metadata, e.g. file size) and *full checks* (doing a fixity analysis on the file and comparing with fixities in the system). This part of the model is linked to the core entity model (Bitstream and Fixity), and to the storage model (Adapter).

Figure 2.6. Integrity check model (storage)



Although integrity check *events* could logically be recorded as event actions (see Section 2.7.1), the volume of checks would be too large, so instead, bitstreams are grouped together into **storage groups** and checks (pending and completed) are recorded against these groups rather than against each bitstream, or against an entity like the CO as would be the case for events. Storage integrity checks are recorded against a storage group and an adapter – one group can be checked against many different adapters.

Storage groups have no meaning beyond integrity checking. Bitstreams that are together in the same storage group may have no logical relation.

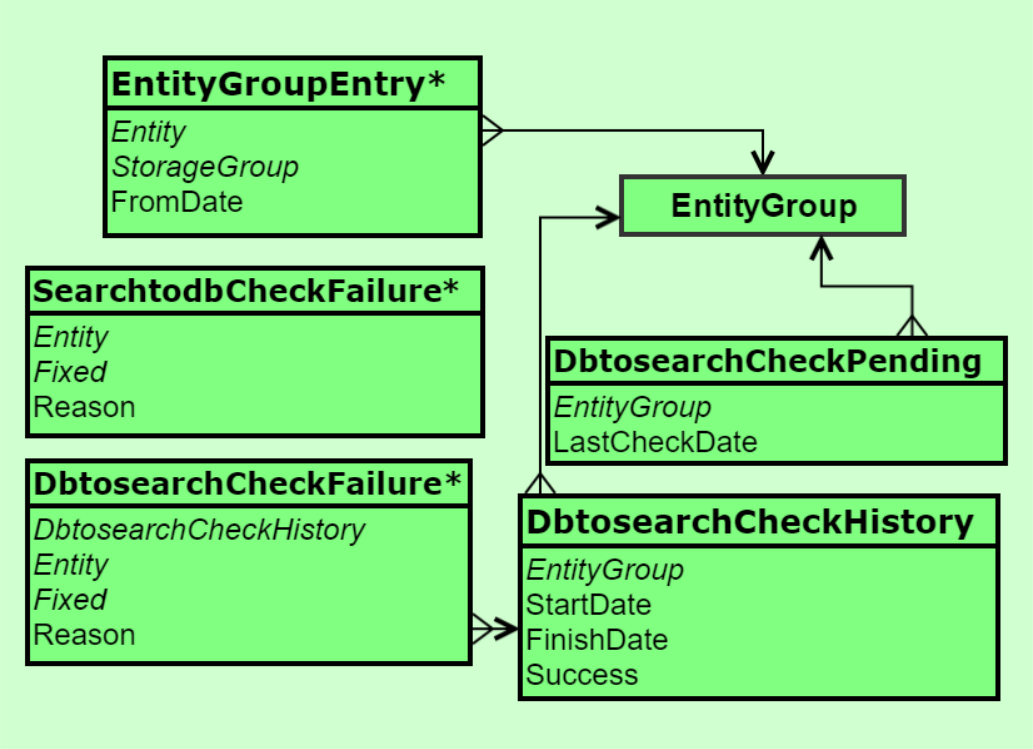
The domain objects for integrity check failures share a common base class (FailedIntegrityCheck) but there is a separate class for the quick and full check failures, and a separate database table for each. Failed full checks reference the fixity for which the value was wrong.

Integrity check configuration is stored in the Settings Data Model and in the settings database.

2.7.3. Integrity Checking (Search)

The search integrity model is similar to the storage integrity model, but because the search index is populated from folders and assets, not bitstreams, and it isn't dependent on storage location, it is simpler and linked only to entities, not to the storage model.

Figure 2.7. Integrity check model (search). Starred objects refer to an entity (SO/IO)

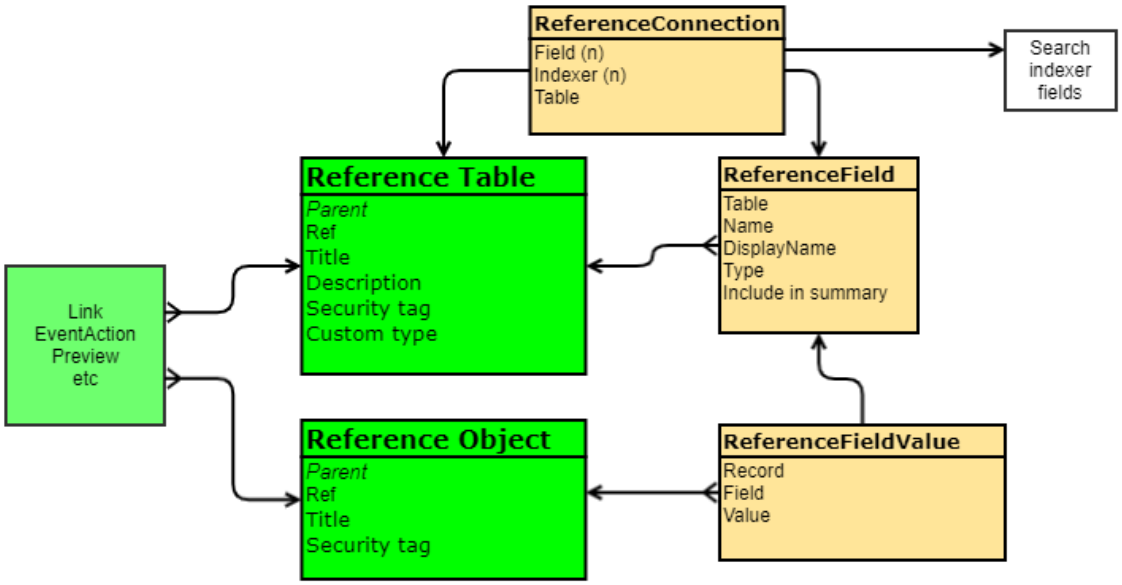


Entities are grouped together into integrity checking **entity groups**. Once again, these groups have no logical meaning outside search integrity checking.

Only database to search integrity checks store a full history. Search to database checks only record failures.

Chapter 3. Reference Metadata

Reference metadata (available from Data Management in the application) is represented as a combination of entity objects and objects specific to this part of the model. Because reference metadata tables are configurable by users at runtime, they do not map to fixed tables in the database, but instead to a set of table/field/value triples linked to fields.

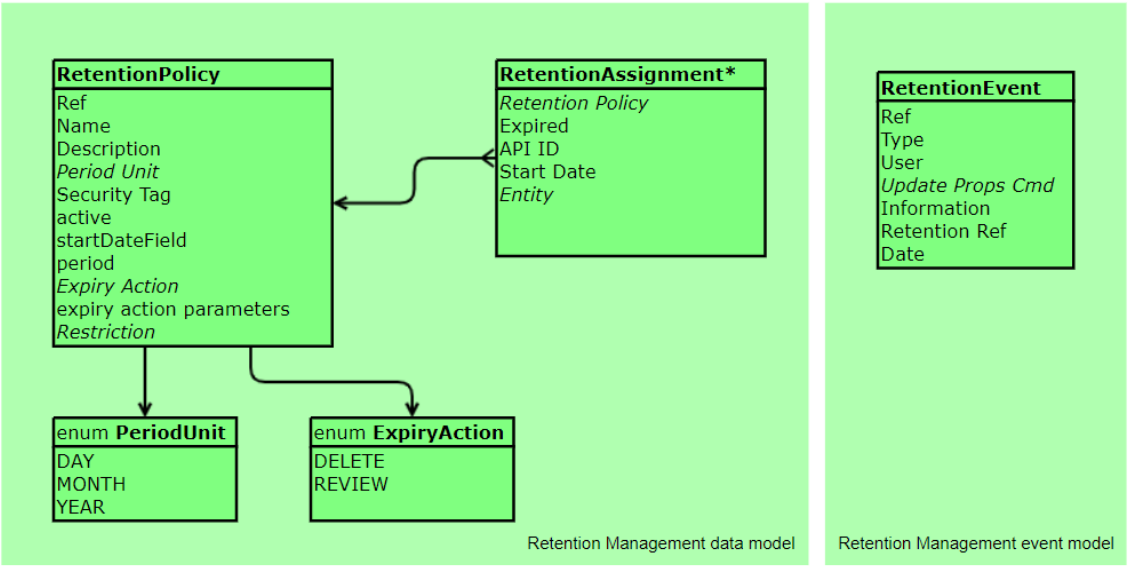


Reference tables (RTs) and *rows* (Reference Objects, or ROs) are Entities, with the same capabilities in the model as SO/IO/CO (e.g. security, events, metadata), although most of this is not available through the application. Reference Fields and Connections define the *columns* and the link to entity metadata, as configured through the application.

Chapter 4. Retention Management

Retention management is a new feature that has been added to Preservica v6.2.1. It allows system managers to create retention policies in the system, which are a set of rules and actions that can be applied to assets (IOs). A policy can either define an action that is going to be taken on the asset at the end of a set period of time, or a restriction that applies to the asset for a set period of time. A policy that only enforces a restriction on an asset, can also be unbounded in time. A policy that is bounded in time can both define an action *and* a restriction.

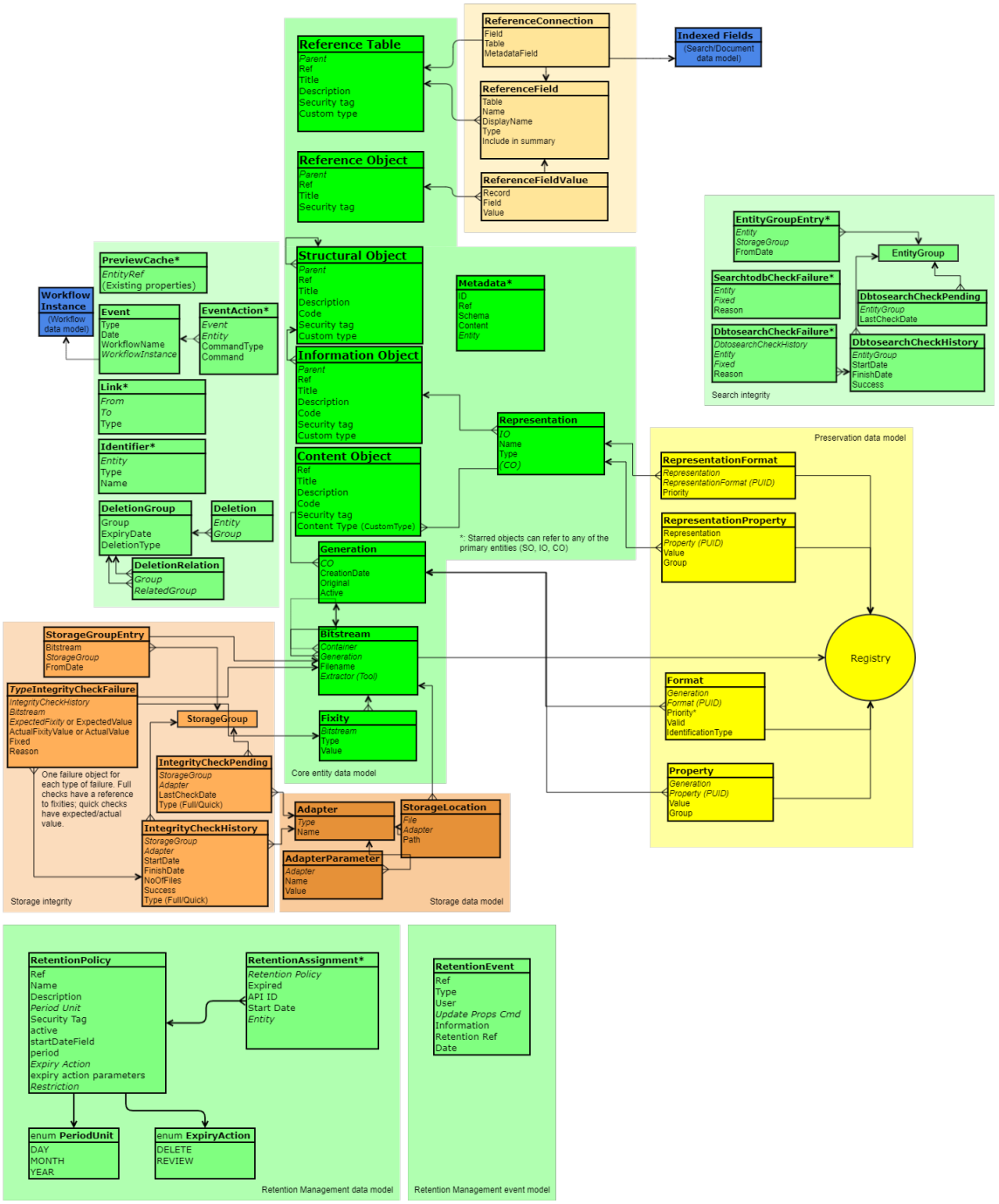
Figure 4.1. Retention Management model. Starred objects refer to an entity (only IOs as of v6.2.1)



Once a policy has been created, it can be assigned to an entity. As of v6.2.1, an entity can only have one policy assigned at a time.

Retention policy events are independent of core entity events and exist in a separate table.

4.1. Reference: Complete Logical Entity Model



Chapter 5. The Database Entity Data Model

The physical representation of the data model in a relational database is very similar to the logical model. Each object in the logical model has a database table to represent it, with the following adjustments:

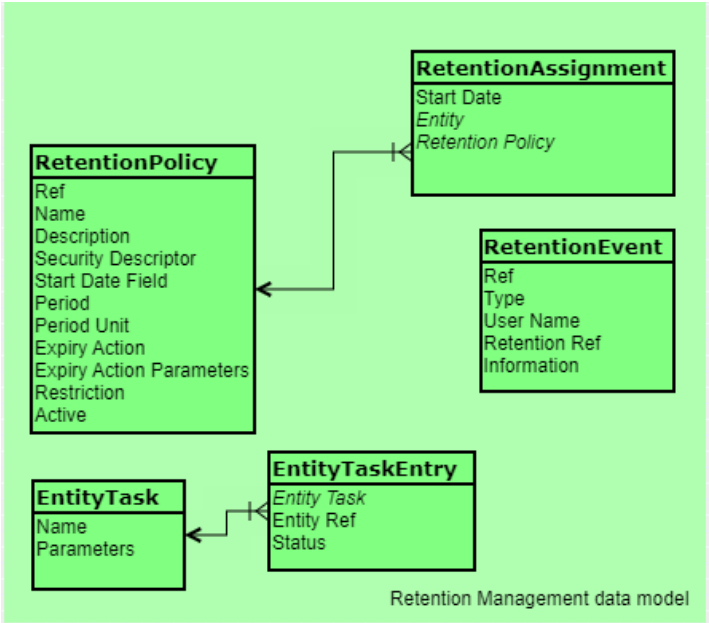
- All metadata entities (SO, IO and CO; and RT and RO) are stored in a single **Entity** table, with a Type column to differentiate between them. This allows objects which reference entities (links, events etc) to have a foreign key relationship to this table.
- Where there is a many-to-many relationship (generation-bitstream) or a one-to-many referencing an entity (representation-CO), in a traditional relational database, there is an extra join table to represent the link.
- All tables (except the join tables) have a numeric ID column, created and last modified dates, even if they are not used in the logical model.
- Table and property names use underscores between words, rather than capitalisation. For example, the database table for *StorageLocation* is named *storage_location*, and the *SecurityDescriptor* property of entities is in a column named *security_descriptor*.

5.1. Metadata Fragments

Metadata fragment content is stored outside the main entity database, either in a file system location, or in a separate database. If you choose to store it in a database, that database will have one table (*metadata_clob*) containing the fragment content and a ref. The *location* property of the Metadata entry in the main database will refer to the ref in the metadata database.

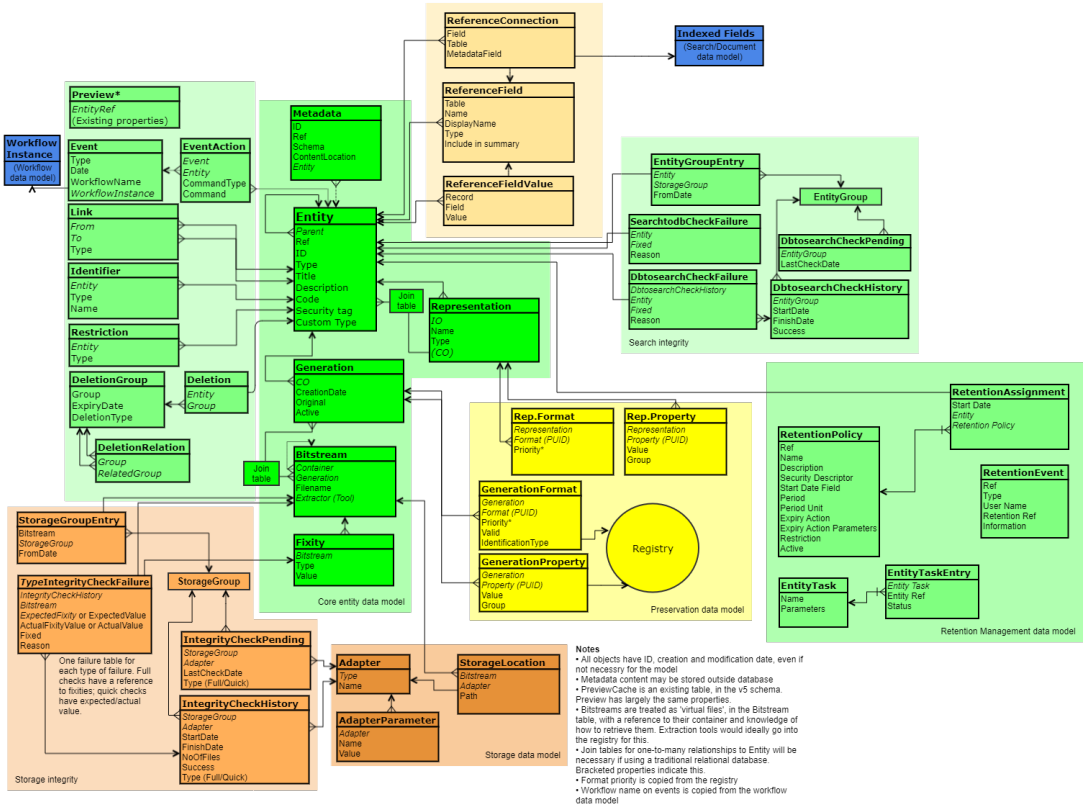
5.2. Retention Management

The ER model for retention management is a relatively simple mapping from the logical model. The only thing to note is the newly added table Entity Task to hold information about *actions on entities*, which are commands waiting to be executed on an entity.



Retention Management data model

5.3. Reference: Complete Database Entity Model



Chapter 6. XIP (v6) as XML

The core entity model, some of the associated entities, and the preservation model, are also represented in an XML schema, which is available from the Preservica user portal.

The XIP schema has a top level XIP object, which contains other entities in the data model. The relationships between objects are the same as in the logical data model, but links between entities are specified by refs as simple strings, replacing the relevant element, and where objects refer to external content (metadata fragments and content files), those links are specified differently.

XIP objects must be in the XIP container in the following order. There are no list wrapper elements at the top level. All of these elements are optional, though in general a useful XIP document will need to contain at least an IO, representation, CO, generation and bitstream.

- `<StructuralObject>`: containing the simple XIP properties. The Parent subelement is the ref of the parent folder, if there is one. Parents may be objects already in the system, or specified elsewhere in the document.
- `<InformationObject>`: similar to SOs
- `<Representation>`: references its parent IO by ref. The representation also contains a container element, `ContentObjects`, containing a list of `ContentObject` elements, each of which is a simple string containing a ref to a CO specified elsewhere in the document. Representations may also contain `RepresentationFormats` and `RepresentationProperties` containers, containing characterisation information; on ingest this is populated by the Characterise step of the ingest workflow.
- `<ContentObject>`: similar to SOs and IOs. Note that the parent of a CO is the IO which owns the representation containing that CO.
- `<Generation>`. The *original* and *active* properties are attributes, and the CO property is a ref. Generations have a `Bitstreams` container element, containing a list of `Bitstream` elements, each of which is a simple string containing a file path (physical location plus file name) to a bitstream elsewhere in the document. Note that this means you can't have two bitstreams with the same path. Generations may also contain `Formats` and `Properties` containers, containing characterisation information; on ingest this is populated by the Characterise step of the ingest workflow.
- `<Bitstream>`: represents a bitstream in the system, but also points to the physical location of the relevant file. At ingest time, the path is relative to the content directory of the ingest package; at export time, it is relative to the directory of each XIP document. The file name is what is saved in the entity data model; the physical location allows the file on disk to be matched to the XIP entity. Bitstreams also contain a `Fixities` container, with a list of `Fixity` elements specifying an algorithm/value pair. A bitstream may have a `Container` element, which references another `Bitstream` in which the bitstream is embedded. In this case, the bitstream will also have an `ExtractionTool` element, which is a label identifying the tool used to extract the bitstream from its `Container`, and an `ExtractionToolParams` element, which will be a JSON string listing any parameters the `ExtractionTool` requires (e.g. a filename or byte range specifiers).
- `<Identifier>`, with the type and value, and a reference to the entity
- `<Link>`, giving the type, and with the *From* and *To* elements being refs to entities.
- `<Restriction>`, with the type, and a reference to the entity
- `<Metadata>`: see below
- `<EventAction>`: when running an export with audit information, the event actions associated with the entities in the export will be included; see the Standard Workflows document for more detail. Unlike the other parts of XIP, this is not a complete replication of the logical event model.

6.1. Metadata Fragments in XIP

In the physical model, once metadata fragments have entered the system, their content is stored separately from information about them. However, the `<Metadata>` element in XIP contains both:

```
<Metadata schemaUri="http://preservica.com/ExtendedXIP/v6.0">
        <Ref>f2652748-f5d9-4256-aac8-da9d010d185b</Ref>
        <Entity>381950bd-d474-4ffc-b772-17234ee1d4e9</Entity>
        <Content>
                <ExtendedXIP xmlns="http://preservica.com/ExtendedXIP/v6.0">
                        <DigitalSurrogate>false</DigitalSurrogate>
                        <CoverageFrom>2019-06-28T15:17:20.669+01:00</
CoverageFrom>
                        <CoverageTo>2019-06-28T15:17:20.669+01:00</
CoverageTo>
                </ExtendedXIP>
        </Content>
</Metadata>
```

The ref and schema URI match what is stored in the Metadata record in the logical model. The content is stored in the metadata content store. Content may contain any valid XML, but it must be XML metadata, contain a single top level node (the `ExtendedXIP` in this example) and the XML namespace used in that node should match the schema URI used on the `Metadata` element.

6.2. Example: Simple XIP for Ingest

This example creates a new top level folder, with a single asset inside it, referencing a single image, which is found in the root of the package's content directory.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XIP xmlns="http://preservica.com/XIP/v6.2">
    <StructuralObject>
        <Ref>381950bd-d474-4ffc-b772-17234ee1d4e9</Ref>
        <Title>Single_image</Title>
        <Description>Single_image</Description>
        <SecurityTag>open</SecurityTag>
    </StructuralObject>
    <InformationObject>
        <Ref>eb660926-fda0-40ae-a878-e82af22ce636</Ref>
        <Title>R</Title>
        <SecurityTag>open</SecurityTag>
        <Parent>381950bd-d474-4ffc-b772-17234ee1d4e9</Parent>
    </InformationObject>
    <Representation>
        <InformationObject>eb660926-fda0-40ae-a878-e82af22ce636</
InformationObject>
        <Name>Preservation-1</Name>
        <Type>Preservation</Type>
        <ContentObjects>
            <ContentObject>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</
ContentObject>
        </ContentObjects>
    </Representation>
    <ContentObject>
        <Ref>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</Ref>
```

```
        <Title>R</Title>
        <SecurityTag>open</SecurityTag>
        <Parent>eb660926-fda0-40ae-a878-e82af22ce636</Parent>
    </ContentObject>
    <Generation original="true" active="true">
        <ContentObject>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</ContentObject>
        <EffectiveDate>2019-06-28T15:17:24.804+01:00</EffectiveDate>
        <Bitstreams>
            <Bitstream>/R.png</Bitstream>
        </Bitstreams>
        <Formats/>
        <Properties/>
    </Generation>
    <Bitstream>
        <Filename>R.png</Filename>
        <FileSize>1185</FileSize>
        <PhysicalLocation></PhysicalLocation>
        <Fixities>
            <Fixity>
                <FixityAlgorithmRef>MD5</FixityAlgorithmRef>
                <FixityValue>6b2478ceced5cabbc5fe69b408573249</FixityValue>
            </Fixity>
        </Fixities>
    </Bitstream>
    <Metadata schemaUri="http://preservica.com/ExtendedXIP/v6.0">
        <Ref>f2652748-f5d9-4256-aac8-da9d010d185b</Ref>
        <Entity>381950bd-d474-4ffc-b772-17234ee1d4e9</Entity>
        <Content>
            <ExtendedXIP xmlns="http://preservica.com/ExtendedXIP/v6.0">
                <DigitalSurrogate>false</DigitalSurrogate>
                <CoverageFrom>2019-06-28T15:17:20.669+01:00</CoverageFrom>
                <CoverageTo>2019-06-28T15:17:20.669+01:00</CoverageTo>
            </ExtendedXIP>
        </Content>
    </Metadata>
</XIP>
```

6.3. Extended XIP

We also define an *Extended XIP* metadata schema. This schema is a list of optional elements which can be added to an entity for extra information. Extended XIP is treated as any other descriptive metadata fragment. The valid properties are:

- DigitalSurrogate (boolean): whether the entity is a digital representation of a physical object held outside the archive. If set to true, this will also cause Explorer to show the digital surrogate overlay in Browse view.
- CoverageFrom and CoverageTo (datetime): can be used to indicate a range of time that the entity covers. Note that these fields must be set to valid `xs:dateTime` values i.e. an ISO-8601 timestamp.
- Classification (string): arbitrary text that can be used to label an entity. Preservica doesn't do anything special with this field, but you can use it in searches and manage records externally via the APIs.

6.4. Additional Information on Export

When you run an Export workflow,you may request audit information in the metadata, which will populate the EventAction elements. In addition, exported content will include the characterisation information (Formats and Properties) on any included generations.

This example is an export of a structural object containing a single asset.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XIP xmlns="http://preservica.com/XIP/v6.2">
    <InformationObject>
        <Ref>eb660926-fda0-40ae-a878-e82af22ce636</Ref>
        <Title>R</Title>
        <SecurityTag>open</SecurityTag>
        <Parent>381950bd-d474-4ffc-b772-17234ee1d4e9</Parent>
    </InformationObject>
    <Representation>
        <InformationObject>eb660926-fda0-40ae-a878-e82af22ce636</
InformationObject>
        <Name>Preservation-1</Name>
        <Type>Preservation</Type>
        <ContentObjects>
            <ContentObject>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</
ContentObject>
        </ContentObjects>
                <RepresentationFormats/>
                <RepresentationProperties/>
    </Representation>
    <ContentObject>
        <Ref>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</Ref>
        <Title>R</Title>
        <SecurityTag>open</SecurityTag>
        <CustomType>image</CustomType>
        <Parent>eb660926-fda0-40ae-a878-e82af22ce636</Parent>
    </ContentObject>
    <Generation original="true" active="true">
        <ContentObject>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</ContentObject>
        <FormatGroup>png</FormatGroup>
        <EffectiveDate>2019-06-28T15:17:25+01:00</EffectiveDate>
        <Bitstreams>
            <Bitstream>Representation_Preservation_1/R/Generation/R.png</
Bitstream>
        </Bitstreams>
        <Formats>
            <Format valid="true">
                <PUID>fmt/12</PUID>
                <Priority>1</Priority>
                <IdentificationMethod>Signature</IdentificationMethod>
                <FormatName>Portable Network Graphics</FormatName>
                <FormatVersion>1.1</FormatVersion>
            </Format>
            <Format valid="false">
                <PUID>fmt/11</PUID>
                <Priority>2</Priority>
                <IdentificationMethod>Signature</IdentificationMethod>
                <FormatName>Portable Network Graphics</FormatName>
                <FormatVersion>1.0</FormatVersion>
            </Format>
        </Formats>
        <Properties>
            <Property>
                <PUID>prp/3</PUID>
```

```
                <PropertyName>Image Width</PropertyName>
                <Value>64</Value>
            </Property>
            <Property>
                <PUID>prp/4</PUID>
                <PropertyName>Image Height</PropertyName>
                <Value>64</Value>
            </Property>
            <Property>
                <PUID>prp/9</PUID>
                <PropertyName>Bits Per Sample</PropertyName>
                <Value>8</Value>
            </Property>
            <Property>
                <PUID>prp/5</PUID>
                <PropertyName>Colour Space</PropertyName>
                <Value>RGB</Value>
            </Property>
            <Property>
                <PUID>prp/7</PUID>
                <PropertyName>X Sampling Frequency</PropertyName>
                <Value>3779</Value>
            </Property>
            <Property>
                <PUID>prp/1</PUID>
                <PropertyName>Compression Type</PropertyName>
                <Value>Deflate</Value>
            </Property>
            <Property>
                <PUID>prp/8</PUID>
                <PropertyName>Y Sampling Frequency</PropertyName>
                <Value>3779</Value>
            </Property>
            <Property>
                <PUID>prp/6</PUID>
                <PropertyName>Sampling Frequency Unit</PropertyName>
                <Value>Metre</Value>
            </Property>
        </Properties>
    </Generation>
    <Bitstream>
        <Filename>R.png</Filename>
        <FileSize>1185</FileSize>
        <PhysicalLocation>Representation_Preservation_1/R/Generation</
PhysicalLocation>
        <Fixities>
            <Fixity>
                <FixityAlgorithmRef>MD5</FixityAlgorithmRef>
                <FixityValue>6b2478ceced5cabbc5fe69b408573249</FixityValue>
            </Fixity>
        </Fixities>
    </Bitstream>
    <EventAction commandType="command_create">
        <Event type="Ingest">
            <User>admin</User>
            <WorkflowName>Ingest into drop folder</WorkflowName>
```

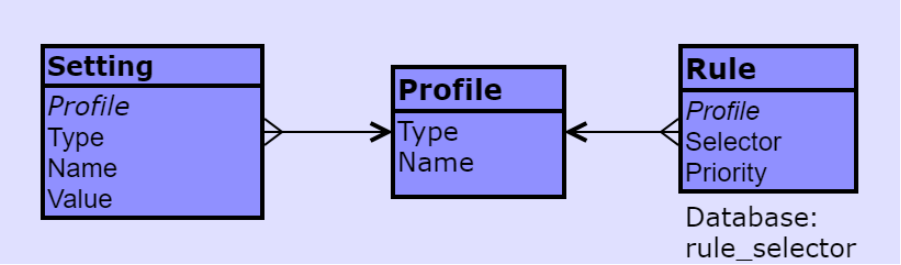
```
        </Event>
        <Entity>eb660926-fda0-40ae-a878-e82af22ce636</Entity>
    </EventAction>
    <EventAction commandType="command_characterisation">
        <Event type="Characterise">
            <User>admin</User>
            <WorkflowName>Ingest into drop folder</WorkflowName>
        </Event>
        <Entity>eb660926-fda0-40ae-a878-e82af22ce636</Entity>
        <SerialisedCommand>{"generations":["3b3f7f6b-fb8a-4914-9c56-
c975213f6ea1-0"],"toolName":null}</SerialisedCommand>
    </EventAction>
    <EventAction commandType="command_create">
        <Event type="Ingest">
            <User>admin</User>
            <WorkflowName>Ingest into drop folder</WorkflowName>
        </Event>
        <Entity>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</Entity>
    </EventAction>
    <EventAction commandType="command_identification">
        <Event type="Characterise">
            <User>admin</User>
            <WorkflowName>Ingest into drop folder</WorkflowName>
        </Event>
        <Entity>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</Entity>
        <SerialisedCommand>{"generations":["3b3f7f6b-fb8a-4914-9c56-
c975213f6ea1-0"],"toolName":"DROID - Signature File V17504"}</
SerialisedCommand>
    </EventAction>
    <EventAction commandType="command_validation">
        <Event type="Characterise">
            <User>admin</User>
            <WorkflowName>Ingest into drop folder</WorkflowName>
        </Event>
        <Entity>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</Entity>
        <SerialisedCommand>{"generations":["3b3f7f6b-fb8a-4914-9c56-
c975213f6ea1-0"],"toolName":"Jhove PNG Module v1.20.1","outcome":"true"}</
SerialisedCommand>
    </EventAction>
    <EventAction commandType="command_property_etraction">
        <Event type="Characterise">
            <User>admin</User>
            <WorkflowName>Ingest into drop folder</WorkflowName>
        </Event>
        <Entity>3b3f7f6b-fb8a-4914-9c56-c975213f6ea1</Entity>
        <SerialisedCommand>{"generations":["3b3f7f6b-fb8a-4914-9c56-
c975213f6ea1-0"],"toolName":"Jhove PNG Module v1.20.1"}</SerialisedCommand>
    </EventAction>
</XIP>
```

Chapter 7. The Settings Data Model

7.1. Settings and Policy

Preservica has a general settings and policy model which is applied to various areas of the system (storage, migration, OCR). In this model, settings are grouped together into a profile, and then a profile is selected based on evaluating rules against content. The difference between *settings* and *policy* is that a policy is applied in the background on all content, whereas a setting is only applied on new content or on request, but otherwise they are the same. For more information, see the System Administration Guide.

This leads to a model where the profile is the central object, containing settings and being referenced by rules:



The value of both a Setting and a Rule are complex and extensible, so they are stored as a string and relevant domain objects created when needed:

7.1.1. Rule Selectors

A **CompositeSelector** is created for a Rule when needed. The rule selector may have up to four parts:

- A **HierarchySelector**, with an entity reference.
- A **FormatSelector**, with a list of format specifiers (in 6.0, PUIDs).
- A **RepresentationSelector**, with a representation type.
- A **SecurityDescriptorSelector**, with a list of security tags.

7.1.2. Settings

Settings are registered together as a *profile group*, which represents the set of settings that make up a particular type of profile. For example, the OCR profile group contains the OCR target and language settings. Profile groups are not stored in the database, but registered within the application. The definition of a setting within a profile group is a *setting group*, which defines the setting's type and any additional information related to the setting.

A Setting can be of several types, each of which is encoded differently:

- **SHORT_STRING** and **LONG_STRING** are stored as text. A short string must be 16 characters or less; a long string up to 128.
- **DROPDOWN** is stored as text, but must be selected from a list of allowed values specified in the group.
- **MULTI_LIST** is stored as a comma separated list of values. What the values mean is specified in the group and in the relevant settings page.

- **MIGRATION_RULE** records a JSON object containing an active flag, and a map of format specifiers (i.e. PUIDs) to migration rule IDs. The ID is the GUID of the rule as specified in the Registry.

7.2. Integrity Settings

The settings model also contains configuration settings which aren't stored on Settings pages, controlling how integrity checking is set up. This data is stored in two independent domain objects: **SearchIntegrityCheckConfig** and **StorageIntegrityCheckConfig** (database tables search_integrity_check and storage_integrity_check). These objects store the information configured on the Storage Integrity page (see the System Administration Guide) and also record when each check was last run.

Note that only the configuration is stored here, information about integrity check results, and grouping of entities for running the checks, is in the entity model (see Section 2.7.2).

Chapter 8. Process and Security

Preservica also stores non-entity data in other databases. The Process/Security database stores information about workflows (both configuration, i.e. definitions and contexts, and execution, i.e. instances, steps, job information etc), security (roles, permissions, user sessions, access token management etc) and some administration and system configuration.

*

www.preservica.com

Reference: git/doc/ADD/LDM | Issue: 6.9.0

Preservica, 32 The Quadrant, Abingdon Science Park, Abingdon, Oxfordshire, OX14 3YS, UK info@preservica.com - preservica.com