Using Media Manifest, File Manifest and Avails for fFile Delivery (Best Practices)

Showing changes from v1.0

CONTENTS

REVISION HISTORY

Version	Date	Description
1.0	January 3, 2015	Initial publication
1.1	June 12, 2015	Updated to reflect changes in Manifest v1.4 Updated to reflect changes in Avails v2.0 Country-specific trailers Improved TV practices Profile-based Track Selection Various other improvements

1 INTRODUCTION

This document describes how to use Avails, Common Media Manifest and Common File Manifest when delivering files from a studio (or service provider) to a party that will use that media (retailer, streaming service, etc.)

This is designed to be an explanatory document rather than a formal specification, however practices are defined so they can be adopted.

This document assumes familiarity with the referenced specifications, particular *Common Media Manifest Metadata*.

1.1 Background

The MovieLabs Common Manifest allows complex user experiences to be assembled from individual assets, e.g. a video track, audio track, etc.

1.2 Document Organization

This document is organized as follows:

1. Introduction—Provides background, scope and conventions
2. Delivery Model – Describes the assumed models for delivering content
3. Identifiers – Describes the identifiers used, how they are formatted and the use of EIDR IDs in this model
4. Connecting Objects – Following the discussion of identifiers, this section describes how Avails, Experiences and other objects are related
5. Avails – Provides information on how to encode Avails
6. Manifest Construction – Describes how to build and use a Media Manifest
7. File Delivery – Describes how to build and use a File Manifest
8. Other Encoding Rules – Provides additional guidance on using other objects (e.g., Regions and Territories).

1.3 Document Naming and Conventions

This document uses conventions as defined in [CM]. This is a less formal document, so strict conventions may not expressly apply in all cases.

1.4 Normative References

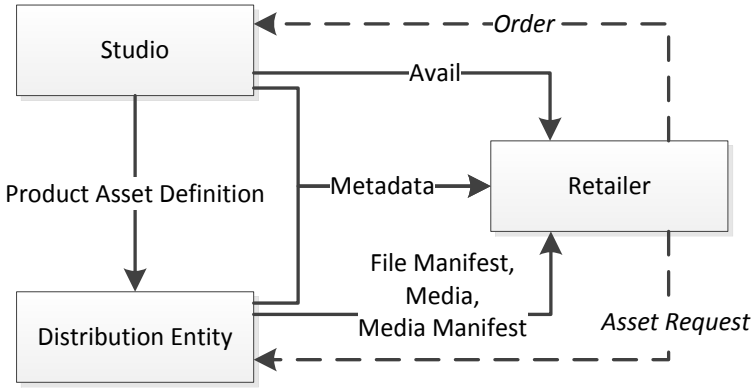
[CM]	Common Metadata, TR-META-CM, www.movielabs.com/md/md
[Manifest]	MovieLabs Common Media Manifest Metadata v1.~~1~~4, TR-META-MMM, www.movielabs.com/md/manifest
[Avail]	EMA Content Availability Data (Avails), TR-META-AVAIL, www.movielabs.com/md/avails
[MEC]	Media Entertainment Core, TR-META-MEC, www.movielabs.com/md/mec
[EIDR-UG]	EIDR 2.0 Registry User's Guide, eidr.org/technology/
[EIDR-ID]	EIDR ID Format, eidr.org/technology/

1.5 Informative References

[DOI]	Digital Object Identifier (DOI), www.doi.org
[XML-C]	Canonical XML, Version 1.0. http://www.w3.org/TR/xml-c14n
[ISO26324]	ISO 26324:2012, *Information and documentation -- Digital object identifier system*
[EIDR-V]	How to Use EIDR to Identify Versions for Distribution Purposes: Edits, Languages and Regional Releases (FAQ), eidr.org/technology

2 DELIVERY MODEL

The delivery model assumed by this document is as follows:



2.1 Roles and Data Objects

This model has the following roles. Note that these are roles and do not necessarily align exactly with corporate entities.

- Studio – Entity that defines the product and the business rules around the product.
- Distribution Entity – generating media files, Media Manifest and File Manifest. This function is often provided by a post-production organization.
- Retailer – Consumes the various pieces as part of offering an experience to the consumer. Although the term Retailer is used, it does not necessarily imply selling the product.

The model also addresses various data objects and messages. Many of these are described in referenced specifications:

- Product Asset Definition – This is the definition of the various components (abstractly) that together create an offering. In this document the collection of assets is referred to as a Logical Asset
- Avail – Data defining assets (abstractly) and business terms; such as, when and where the assets can play, and what pricing tier is used. [Avail]
- Order – A Retailer will order an Avail indicating it wishes to offer the Assets in accordance with the Terms.
- Asset Request – A Retailer will request assets from the Distribution Entity.
- File Manifest – Data describing a set of files that are part of a delivery. Defined in [Manifest].
- Media Manifest – Definition of how various media assets are connected to provide an experience to a user. Defined in [Manifest].
- Metadata – Consumer-facing description of media assets. This is preferably based on the Media Entertainment Core (MEC) [MEC].

2.2 Distribution Workflow

This section describes the workflow between the Studio/Post-Production and the Retailer.

The Studio in conjunction with Post-Production determines which assets are available. The Retailer requests delivery of assets. Post-Production then delivers the assets. There are various variations on this model. The model supports partial deliveries, issue handling, and updates.

The Media Manifest serves as a bill of materials (BOM), describing the various pieces that need be assembled into the final product.

While the Studio and Post-Production determine the set of assets it wishes to create for a title, Retailers do not necessarily want everything. For example, a Retailer may not want a 2-channel audio track if a 5.1 track is available. The process of culling a delivery is currently outside the scope of this document, and is handled bilaterally between Post-Production and the Retailer. Generally, the Retailer will know what assets it needs because of standard templates for regions based on business terms and technical capabilities. The bilaterally agreed upon deliveries would likely be based on these templates.

Exact usage is determined jointly by some combination of Studio, Retailer and Post-Production.

2.2.1 Standard Delivery

In a Standard Delivery, the studio and post-production facility determine what must be built and prepare everything needed for fulfill an Avail.

Whether the materials are pushed to the Retailer or requested by the Retailer depends upon agreements outside the scope of this document.

The Standard Delivery process is as follows:

- Asset Preparation (not necessarily sequenced with Ordering)
    - The studio provides instructions for the post-production facility needed to construct the appropriate deliverables to the Retailer
    - Post-production generates
        - Media files
        - Media Manifest
- Ordering
    - The Studio provides an Avail to the Retailer
    - The Retailer receives that Avail and opts to offer that title
    - The Retailer informs the Studio in accordance with their business agreements.
- Fulfillment

- The Retailer orders material based on the A~~vail~~LID. This step could be skipped if the Distribution Entity is otherwise aware that the Retailer is licensed to distribute the assets covered by the Avail.
- The Distribution Entity delivers a File Manifest and a collection of files including the Media Manifest, metadata and media.

In practice the Retailer would have specific delivery requirements that might require additional ordering information. For example, a Retailer might want to use adaptive streaming streams or Common File Format (CFF) files.

The intent is to create and ordering specification. See Annex A for draft ordering requirements.

2.2.1.1 Retailer Criteria

The Retailer may bilaterally establish delivery rules with the Distribution Entity. These may include preferred track types and minimum requirements per region/language.

The Distribution Entity will filter materials that are not required by the Retailer. The Distribution Entity may prioritize asset production based on these criteria. The Distribution Entity will flag deliveries that do not comply with these criteria.

2.2.2 Update flow

This section describes the models for update.

The first model is "Push", where the Distribution Entity sends updates as it deems appropriate. This could happen when a component has been improved or corrected, or when the Distribution Entity knows the Retailer is waiting for something (e.g., a new track). The other model is "Pull" where the Retailer orders updates. These processes are described below.

- Push Model
  - The Distribution Entity generates a File Manifest and updates the Manifest (at least the Inventory)
  - The Distribution Entity sends this manifest and files via normal channels.
  - The Retailer receives the files
- Pull Model
  - Retailer becomes aware that new material is available. For example,
    - Studio informs Retailer that a new component (e.g., next generation video) or revised component is available.
    - The Distribution Entity informs Retailer that a new or revised track is available.

- Retailer decides to obtain that material
- Retailer either requests or receives an update Manifest (based on same ordering criteria as original order)
- The Retailer uses the Media Manifest to determine which files are to obtain the new track.
- Retailer requests files (order)
- (time might pass if files not ready)
- The Distribution Entity generates a File Manifest and delivers files through normal channels.

See Inventory-only Media Manifest Updates below for a description of how to delivery only the Inventory portions of a Media Manifest.

2.2.3 Issue Resolution Delivery

In case issues are found with a delivery, it will be necessary for a Retailer to notify the Distribution Entity that new information needs to be issued. Some examples of these problems include:

- A file is referenced in Media Manifest, but was not in the File Manifest
- A file is referenced in the File Manifest but is not available
- Content is referenced in an Avail, but does not have an associated Experience
- Material is implied in the terms of the Avail (e.g., Russian subs and dubs will be provided), but it is not in the Experience.
- Corrupted or badly formed files are delivered.

The resolution process is as follows:

- Retailer informs the Distribution Entity that one or more files is problematic
  - There is likely human intervention. There needs to be human-readable text indicating problem.
- The Distribution Entity responds, possibly delivering updated files via one of the delivery processes above.

See Inventory-only Media Manifest Updates below for a description of how to deliver only the Inventory portions of a Media Manifest.

2.2.4 Updates

While updates can be performed using the Media Manifest, the most straightforward method is to use MedianInventory and MediaManifestEdit.

2.2.4.1 Inventory-only Media Manifest update

In updates and issue resolution, it is possible that only the Inventory changes. To facilitate delivering the Inventory alone, there is a MediaInventory element. The essential criteria are that all references to the Inventory are the same.

For example, consider an update where a defective track is replaced. The only change is some combination of TrackIdentifier and ContainerReference. The track reference (TrackID) from a Presentation is not affected.

2.2.4.2 Localization Update including tracks (Localization 1)

The most frequent update is the addition of a language. This section describes the rules for performing a localization update. We expect other models may be required, so we are calling this set of rules Localization 1.

There are the following variations on this model:

- Localization 1 – Localization including any language and region information (i.e., metadata, subs and dubs, etc.)
- Localization 1a – Localization including only metadata (e.g., add artwork for a region)

It is important that the entity sending the update can construct identifiers for objects being updated. This requires either prior knowledge of the identifiers, such as having a current version of the manifest, or having the ability to construct identifiers. The ID naming conventions in this document can help in many cases, but may require additional naming convention agreements between parties.

2.2.4.2.1 Localization 1

Localization 1 and its variants involved sending MediaManifest*Edit*. Take care not to use MediaManifest instead.

The following rules instruct how to populate MediaManifestEdit to update Media Manifest using the Localization 1 model:

- MediaManifestEdit@updateNum must reflect an update. That is, it must be greater than all previous @updateNum values. If by prior agreement, the sequence of updates is understood, this can be waived.
- MediaManifestEdit@updateDeliveryType='Localization1'
- Any data referenced by MediaManifestEdit/DeleteObject will be removed prior to adding objects
- MediaManifestEdit/AddObject/Inventory is present.
    - For each language added

        - At least one Audio or Subtitle element must be present. In this model, there is no such thing as a language update without audio or subtitle.
        - All audio and subtitle tracks for the language that are included in a Presentation must be in included in the Inventory
        - All track identifiers must be unique. That is, they must be distinct from each other and must not exist in any previous version of the manifest.
    - Other Audio, Video and Subtitle tracks are included as appropriate. For example, localized ratings or anti-piracy pre-roll video might be required as part of a localization.
    - Image and Interactive are included as appropriate
- MediaManifestEdit/AddObject/Presentations/Presentation must be present
    - A Presentation must be included for all Presentations that include new tracks.
    - To add tracks to an existing Presentation
        - Presentation/@PresentationID must match the Presentation to be updated
        - TrackMetadata/TrackSelectionNumber must match existing Track Metadata
        - VideoTrackReference, AudioTrackReference and SubtitleTrackReference are only included for new tracks.
        - LanguagePair may be added
        - Chapters may not be modified in any way
        - If an object (e.g., a track reference in a Presentation) in the update already exists in the original object, it is ignored.
        - Note: There is currently no way to delete a track
- MediaManifestEdit/AddObject/PlayableSequences/PlayableSequence may be added.
- MediaManifestEdit/AddObject/Experiences/Experience can be added.
    - It must have a unique ExperienceID
    - @updateNum='1'
- MediaManifestEdit/Experiences/Experience can be modified as follows:
    - @ExperienceID must match the Experience being updated
    - @updateNum must reflect an update

- Language and/or Region must be included, these are assumed to be in addition to existing Languages or Regions covered. The following defines how languages are regions are added. The rules are the same for Language and Region.
  - Case 1: All languages/regions are already covered. This is represented by the absence of Language/Region and ExcludedLanguage/ExcludedRegion. Adding a language/region has no effect because they are already implicitly included.
  - Case 2: Language/Region instances exist for other languages/regions. An instance of Language/Region is created for the language/region.
  - Case 3: ExcludedLanguage/ExcludedRegion instance exists for language/region. Adding a language/region requires the instance be removed. By definition if it's not Excluded, it's included.
- If there are new Playable Sequences or Presentations, Audiovisual must be modified or created as follows
  - If @ContentID matches @ContentID in an earlier version of the Experience, this Audiovisual replaces the earlier version
  - If @ContentID does not match @ContentID in an earlier version, Audiovisual is added.

- MediaManifestEdit/AddObjects/BasicMetadata must be included for localizations not already in metadata. Note that if a region is added with a language that is already supported, BasicMetadata might not be needed.
  - @ContentID is required
  - WorkType is required
  - UpdateNum should be included and reflect and update. However, it is expected that updates are provided by uncoordinated parties so it may not be practical to enforce proper UpdateNum usage.
  - LocalizedInfo must exist for each language
    - @default should not be included. Generally, there should already be an instance of LocalizedInfo with @default='true'. If the default LocalizedInfo must be changed, this can be handled through an Instructions-only Special Delivery Condition (2.2.5.3).
  - RatingSet/Rating should be included for covered regions, as applicable
  - People should be updated if localized information is included, such as Job/JobDisplay and Name/DisplayName

- Other elements and attributes need not be included.

- If an update cannot be accomplished within these rules, a complete Media Manifest must be provided.

As an alternative, Media Entertainment Core (MEC) can be used. MEC does not have a means to signal that this delivery is an update so the parties must determine this from context. The following rules apply to updating metadata via MEC in Localization 1:

- BasicMetadata should include all new localizations. Rules are the same as for MediaManifestEdit/AddObject/BasicMetadata above.
- If DigitalAsset is included in metadata delivery it must include new track definitions. Note that DigitalAsset is redundant with Inventory and will likely not be included.
- TitleInternalAlias should be included
- TrackingID must be included as appropriate
- Source should be included
- CompanyDisplayCredit and GroupingEntity should be included if localized data differs from what was previously delivered (of if there is any doubt).

2.2.4.2.2 Localization 1a

Localization 1a is designed to add only metadata to an existing Experience, such as adding region-specific artwork. This will typically result in an updated Experience that includes the new regions. If metadata is included in the Experience, it is necessary to update the Experience and add Inventory for the images. If the Experience references metadata through ContentID (i.e., metadata delivered separately), it is only necessary to include the image. A Localization 1a update can be as simple as an Inventory with an image.

Localization 1a has the same rules as Localization 1, with the following exceptions in MediaManifestEdit:

- @updateDeliveryType='Localization1a'
- MediaManifestEdit/Inventory is present if necessary.
  - Image is included as appropriate
- MediaManifestEdit/Presentations/Presentation is not present
- MediaManifestEdit/PlayableSequences is not present

2.2.5 Special Delivery Conditions

This section describes signaling of special cases. It is essential that both parties be aware of these conventions. Otherwise, unexpected behavior will likely result.

2.2.5.1 Explicit non-delivery

In some cases, the sender needs to signal that a particular (expected) component will not be delivered. For example, a particular subtitle track that would typically be included might be omitted if it were not available.

This is done by including the component, while encoding for the component in question: Inventory/[Audio|Video|Subtitle|Image|Interactive]/TrackReference='NotSupplied'. All component types have TrackReference so any component can be annotated in this manner.

'NotSupplied' is not resolvable as an actual track reference and therefore indicates this condition.

2.2.5.2 Track-level Delivery Exceptions

Additional delivery exceptions can be noted in the Inventory. Each media type (video, audio, etc.) element contains a Private element that contains a sequence of any##other elements. Essentially, you can include anything you want (as long as it's not defined the manifest namespace).

For example, if a track is of substandard quality but is the best track available, one might include:

```
<Private>
   <DeliveryNotice>Track did not pass QC, but this is the best we have available.</DeliveryNotice>
</Private>
```

2.2.5.3 Instructions-only Delivery or Update

In the case where an update is entirely manual, this can be signaled through an automated system using MediaManifestEdit. The following conditions apply

- @ManifestID should be included in the update corresponds with an existing Media Manifest.
- @updateDeliveryType="Instructions"
- @ExtraVersionReference can be used to allow the recipient to have a reference to refer to this set of instructions.
- Description can be populated, if desired
- Instructions must be included. This provides the instructions for the recipient.

2.2.6 New Episode Delivery

Seasons, miniseries and other episodic assets as well as collections can be delivered as new assets become available. For example, if a season pass is sold, episodes would be delivered as they aired.

As described in Section 2.3, Experiences are packaged as their own XML files. Delivering the XML file with the newly released Experience is necessary but not sufficient. It is also necessary

to update the parent object (e.g., season or miniseries) Experience to include the new asset. For example, when a new episode is released the episode Experience is released along with the season Experience.

Each episode is delivered individually. However, since adding an Episode requires updating the season Experience (or series Experience in a miniseries), the season Experience is delivered with the new Episode. This means putting including both the season Experience instances and episode Experience instances in the same ExperienceList.

The construction of the episode Experience is the same regardless of whether each Experience is delivered incrementally or all at once. The parent (season or miniseries) Experience must be updated to include reference to the new asset, both in Experience/ExperienceChild and in ALIDExperienceMap. The season (or miniseries series) Experience/@ExperienceID must be identical in all deliveries of that Experience. Experience/@updateNum must be present and updated.

If the number or nature of the season or series Experience instances change, they must be handled through the ExperienceEdit mechanism. For example, if a season ordering is required for a locale, the Experience structure will change—this cannot be handled by simply sending the new experiences.

Implementers should take care to handle special cases for both miniseries and seasons; and for split (localized) Experiences.

Note that episodes can be updated to add localizations in the same manner as any other asset.

2.3 Packaging Experiences into XML Documents

As a general rule, the best practice is to put a single media entity (e.g., a movie, TV episode, trailer, featurette, etc.) in its own XML document. This allows maximum flexibility for ingestion and incremental deliveries.

2.3.1 General Guidance

All Media Manifest elements relevant to the media entity must be included. This might result in multiple Experience instances in this XML document. For example, there might be Experiences for different regions, languages or windows (pre-order, for-sale). As long as they refer to the same media entity, they should be included in the same XML document.

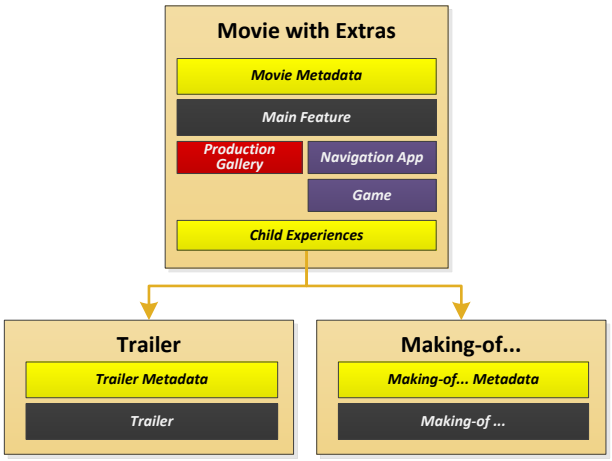
Experiences might reference child Experiences that are contained in other XML documents. It is essential that the cross-references are consistent.

Each XML document has its own ALIDExperienceMap. This references all relevant Experiences in that document. To be clear, if there are child Experiences within the XML document, the parent and children are both referenced. Experiences not in the XML document are not referenced.

2.3.2 Packaging Movies with Trailers and/or Bonus Material

Each feature (i.e., movie), trailer and bonus material should be contained in its own XML document. It should be structured such that the feature's Experience references the trailers, bonus material and other assets.

The following picture illustrates a movie with a trailer and a bonus 'making-of' featurette. This movie also has a production gallery, a navigation application and a game.



2.3.3 Packaging Episodic Experiences

This is best described through an example. Consider the following illustration of a Season. In this example, one XML document would be provided for the Season and one XML document would be provided for each episode. If there was bonus or promotional video (e.g., featurette or trailer), each video's Media Manifest would be in a separate XML document.

The season XML document would have the Experience element, an Inventory/Metadata element, and an ALIDExperienceMap element. The ALIDExperienceMap would contain reference to all existing episode Experiences, even though they are not in that document.

The season Experience must include an ExperienceChild instance for each existing episode. If an episode does not yet exist, it should not be referenced.

Each episode XML document contains a single Experience and all referenced elements. The episode's document, for example, contains any necessary Playable Sequences, Presentations and Inventory elements. It may also include galleries, interactive, Timed Event or other objects relevant to the episode. The episode Experience can reference child Experiences, but these would be contained in separate XML documents.

2.3.3.1 Incremental Episode Deliveries

As mentioned in Section 2.2.6, the season Experience both the updated season Experience and episode Experience are in the same ExperienceList. This means they are delivered as one XML document.

3 IDENTIFIERS

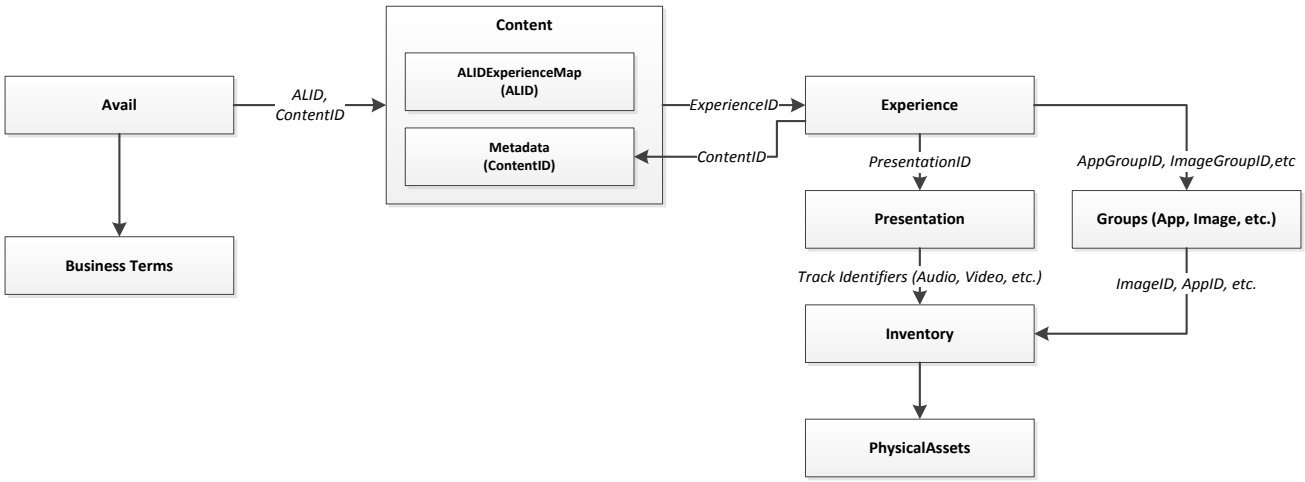
There are multiple identifiers associated with these objects. These are necessary to sustain the information model, but unfortunately they are confusing. In this section, we provide an informal description of these identifiers to provide a better intuitive understanding of what they are.

Most identifiers are specific to what they identify so it is important to use them correctly. Using them correctly will avoid confusion about what is covered by the identified object and will avoid much confusion and many errors.

3.1 Understand IDs of different Types

The identifiers are named somewhat abstractly because when we tried more meaningful names everyone got confused. You'll note that terms like 'product' are rigorously avoided because they are not precise and people have preconceived notions of what they mean. The fact that nobody has a preconceived definition of "logical asset" allows us to define it precisely.

The following diagram shows the relationship between identifiers:



"Content" is at the center. Avails avail Content and Experiences fulfill Content. Logical Asset IDs (ALIDs) and Content IDs identify Content. Although ContentID and ALID are similar they represent distinct concepts and are sometimes different. To understand the distinctions, a careful reading of the sections below is strongly recommended.

Avails reference Content and Business terms. The important concept is that although there may be multiple assets and transactions, *each Avail corresponds with exactly one set of Content*.

Experiences reference Content via ContentID. But, more important to Avails, the ALIDExperienceMap allows one to determine which Experiences can fulfill an Avail.

From the Experiences, one can map the Physical Assets that are used to deliver the experience to the consumer.

3.1.1 Simplifying Identifiers

The identifier structure is designed to handle all cases, but there are simplifications that will simplify identifier creation and interpretation.

~~Avail ID,~~ ContentID, ALID and Experience ID are generally rooted in the same Content so the same base identifier can be used for all. In some cases, not discussed in this document, ExperienceIDs would have their own EIDR ID and therefore have a different base. So, it is important to not make assumptions about base identifiers. It is always possible to tell identifiers apart (e.g., ContentID from ALID) because they are structured differently (e.g., md:cid… vs. md:alid…).

Let's take the following example for a title with the EIDR identifier `md:~~avail~~alid:eidr-x:1012-7947-21D5-9D24-CC5F-H:craigsmovies.com_july_NorthAmerica`. In this example, the ~~AvailID,~~ALID and ContentID ~~and ALID~~ are ~~all~~both based on this EIDR ID.

ID	General Example	Specialized example
A~~vail~~ID	In general, there will be multiple Avails for Content	`md:~~avail~~alid:eidr-x:1012-7947-21D5-9D24-CC5F-H:craigsmovies.com_july_NorthAmerica`
ContentID	`md:cid:eidr-s:1012-7947-21D5-9D24-CC5F-H`	In general, a specialized for is not required.
~~ALID~~	~~`md:alid:eidr-s:1012-7947-21D5-9D24-CC5F-H`~~	~~`md:alid:eidr-x:1012-7947-21D5-9D24-CC5F-H:craigsmovies.com`~~

As there are many Avails for a given title, the EIDR-x form is used with the extension distinguishing the Avail.

As the Avails cover the same Content, there is only one metadata object so a single ContentID will suffice. If there are different metadata sets, then the EIDR-x form could be used, although this is the unusual case because generally a new EIDR ID would be better practice. If considering the EIDR-x format, first consider whether it is the truly same Content or whether there should be a distinct based identifier (e.g., EIDR ID) created. If you end up needing a new EIDR-X for a CID, odds are you should be creating a new base identifier. Recommended practice is to create new base identifies (e.g., EIDR IDs) when there are differences in Content.

If all Avails for this Content maps to the same Experiences, then the EIDR-s form can be used. However, the EIDR-x form gives you the flexibility to create different Experiences for different Avails. One particular use case for this is different Retailers. If each Retailer gets a slightly different experience for the same Avail, even if it is just artwork, then it is necessary to that the Avails use the appropriate identifiers to ensure mapping to the correct Experience. The recommended practice is to use EIDR-x even if you have not identified differences. This provides the flexibility to add variations later.

Note that the base identifier need not be EIDR ID, although EIDR is designed for these scenarios.

3.1.2 Content ID

The Content ID (also referred to as ContentID and CID) is used exclusively as a reference to metadata. As such, it's important for User interface, but it has no actual function in other parts of the workflow, such as Rights management, licensing, distribution, and packaging.

A Content ID can describe an actual work, such as a movie, a TV episode or a short subject, or it can be an object used to group things such as a season, series or franchise.

Everything identified by a Content ID should have Basic Metadata record. Basic Metadata is formally defined in the Media Entertainment Core [MEC].

Keep in mind that Metadata is used by every User Interface including those for processing orders and order resolution.

ContentID covers a particular version of the work, regardless of where that work is released. Metadata can contain multiple instances if localized information, so on ContentID can cover many regions and languages. Only if the edit changes, is a new ContentID required.

3.1.3 Experience ID

The Experience ID identifies an Experience. It's not any more complicated than that, but is essential to have the capability to refer to an Experience.

In this flow, one gets from an Avail to an Experience using the Media Manifest's AvailExperienceMap.

3.1.4 Logical Asset ID (ALID)

The Logical Asset Identifier (ALID) defines the content that a person has Rights to; that is, what is the subject of an Avail. The ALID goes into the Avail and it is used to map Avails to Experiences.

To understand an ALID, it is useful to look at how the ALID is used. When the studio decides the assets in an Avail, an ALID is created. The ALID might cover a single asset (e.g., a movie) or it can be an organized collection of assets (e.g., a season) or an arbitrary collection of assets (e.g., 'movies starring Kirk Douglas'). At the time of the Avail, it is not necessary for the assets associated with the ALID to be fully defined—it can be updated later. However, by the time the assets are to be fulfilled, the ALID must be well defined.

How does one know the assets in the ALID? There are three ways;

- The Avail's AssetList lists the assets. This is intentionally somewhat loose because it is understood that Avails are published before products are fully defined.
- The Media Manifest maps ALIDs to Experiences (ALIDExperienceMap). From there, the Media Manifest (Experience, Presentation, Inventory, etc.) fully defines what is addressed by the ALID

- EIDR unambiguously identifies what an ALID refers to. EIDR definitions correspond to the various assets an Avail must address (i.e., movies, episodes, seasons, arbitrary collections, etc.). We recommend EIDR IDs because of the flexibility and precision. Note that EIDR definitions can be created in rough form, if full information is not available at Avail time, and revised later.

The ALID is the glue that ties everything together.

~~3.1.5 AvailID~~

~~Generally, people will refer to the Avails by title, region and window; but systems need something more precise. An AvailID uniquely defines an Avail. AvailID is a means for systems to refer to Avails. Note that since an AvailID can cover multiple sets of assets (e.g., seasons) and multiple sets of terms (e.g., HD EST, HD VOD, SD EST, SD VOD, etc.) the AvailID can cover quite a bit.~~

~~An Avail Identifier (AvailID) identifies an Avail. More specifically, AvailID uniquely identifies a combination of assets and business terms.~~

~~To~~Furthermore, to be clear, the following describe what an A~~vail~~LID is not:

- An A~~vail~~LID is not necessarily unique to a combination of assets and terms. A studio might assign multiple IDs to the same combination. This might be useful if the same Avail structure is offered to different Retailers.
- An A~~vail~~LID is not a content identifier. In many cases, an Avail will correspond with a single asset. However, the IDs are not interchangeable. Asset is fundamentally different from Asset+Terms. Put differently, an Avail uses Content IDs to refer to items that is being made available.

It is also important to unambiguously identify the content referred to by the Avail, even if it is not fully defined at Avail time. This is what the Distribution Entity must deliver and what the Retailer may offer. The identifier that refers to the collection of assets associated with an Avail is ~~a Logical Asset Identifier (ALID).~~an ALID.

An ~~Avail identifier~~ALID might be used by the studio or various service providers partnering with the studio so Avails must be identified in a globally unique manner. Consequently, the ~~AvailID must~~ALID should be defined globally unique. ~~In most cases AvailID and ALID will be based on the same EIDR.~~ The EIDR-x form ~~can~~should be used to distinguish between the different Avails for the same content.

3.1.5 Identifying Avails

Generally, people will refer to the Avails by title, region and window; but systems need something more precise. We have established the convention of requiring uniqueness for the combination of ALID and Licensor.

An ALID can cover correspond with individual assets or sets of assets (e.g., seasons); as well as multiple sets of terms (e.g., HD EST, HD VOD, SD EST, SD VOD, etc.). In the context of an Avail in combination with Licensor, the ALID uniquely identifies a combination of assets and business terms.

For example, when Avails are updated ~~the AvailID~~(within the context of a Licensor) the ALID is used to know that a particular Avail is to be updated (i.e., the update A~~vail~~LID matches the original A~~vail~~LID). The Avail may be updated, but the ID remains constant.

3.1.6 Track, Image and Interactive IDs

Asset Physical Identifier (APID) uniquely identifies each digital asset.

The Physical Asset ID is more commonly referred to as an APID. In general, the terms Physical Asset and Digital Asset are synonymous, and you will see the term Digital Asset used more commonly in the specifications.

It's important to understand the following

- An APID does not correspond to a specific, single file. A Container is a Container regardless of whether it's in a file by itself, part of a ZIP file or packaged in some other way.
- It is a matter of convention what constitutes a new encoding and requires a new APID. Typically, in a Media Manifest, APID used for tracks will remain constant regardless of encoding. For example, a TrackID will generally refer to the same essence regardless of encoding. That way, a track can be updated without having to update a Presentation.
- APIDs used for files should be unique to a given file. If the file changes (e.g., the encoding changes or certain data in the file change), then a new APID is required.

3.1.7 ID Format

This document defines identifier formats as a best practice. These are not mandated in the referenced specifications, but best practice requires consistency. It may at some point make sense to move these to their respective specifications. Note that only the format is defined here. Use of these identifiers is defined elsewhere in this document.

Identifiers in the document use the Common Metadata [CM], Section 2.1 identifier format <MDID> where

<MDID> ::= "md:"<type> ":"<scheme>":"<SSID>

<scheme> and <SSID> are defined in [CM], Section 2.1.

<type> is defined as follows:

ID	<type>	Constrained form
PackageID	packageid	"md:packageid:"<scheme>":"<SSID>

~~AvailID~~	~~availid~~	~~"md:availid:"<scheme>":"<SSID>~~
ProductID or ALID	alid	"md:alid:"<scheme>":"<SSID>
PresentationID	presentationid	"md:presentationid:"<scheme>":"<SSID>
Experience	experiencedid	"md:experienceid:"<scheme>":"<SSID>

~~ProductID~~

TransactionID	transationid	"md:transactionid:"<scheme>":"<SSID>

ProductID is sometimes used in lieu of ALID (e.g., in the Avails spreadsheet). It is a Logical Asset identifier and consequently uses the ALID format.

The following additional naming conventions are used optionally for identifiers in Media Manifest:

ID	<type>	Constrained form
App Group	appgroupid	"md:appgroupid:"<scheme>":"<SSID>
Picture Group	picturegroupid	"md:picturegroupid:"<scheme>":"<SSID>
Playable Sequence	playablesequence	"md:playablesequence:"<scheme>":"<SSID>
Video Track ID	vidtrackid	"md:vidtrackid:"<scheme>":"<SSID>
Audio Track ID	audtrackid	"md:audtrackid:"<scheme>":"<SSID>
Subtitle Track ID	subtrackid	"md:subtrackid:"<scheme>":"<SSID>
Interactive Track ID	interactiveid	"md:interactiveid:"<scheme>":"<SSID>
Image ID	imageid	"md:imageid:"<scheme>":"<SSID>
Picture ID	pictureid	"md:pictureid:"<scheme>":"<SSID>
Gallery ID	galleryid	"md:galleryid:"<scheme>":"<SSID>

3.2 Using EIDR IDs

Information on using EIDR IDs can be found in the references [EIDR-UG], [EIDR-ID] and [EIDR-V]. Additional information is provided in this section.

3.2.1 EIDR Format (EIDR-s and EIDR-x)

EIDR ~~is~~IDs are based on the Digital Object Identifier (DOI), standardized as ISO 26324 [ISO26324] and described [DOI]. A full EIDR ID looks something like this: `10.5240/1012-7947-21D5-9D24-CC5F-H`. It can be converted to a resolvable form to obtain metadata: `https://resolve.eidr.org/EIDR/object/10.5240/1012-7947-21D5-9D24-CC5F-H`. However, the Common Metadata identifier format uses URN format which requires percent encoding for certain characters such as slash ('/' = '%2f'). So, it's easier to use a form of EIDRs that does not include the "10.5240/" prefix. The forms relevant here are EIDR-S (for 'short') and EIDR-X (for 'eXtended'). These and other forms are defined in [EIDR-ID].

These forms are as follows:

"md:~~avail~~alid:eidr-s:"<EIDR suffix>

"md:~~avail~~alid:eidr-x:"<EIDR suffix>":"<extension>

For example,

```
md:availalid:eidr-s:1012-7947-21D5-9D24-CC5F-H
md:availalid:eidr-x:1012-7947-21D5-9D24-CC5F-
H:craigsmovies.com_july_NorthAmerica
```

The short form is used to express an EIDR ID with no other information. The extended form is used when identifiers are derived from the EIDR ID, but additional information is required to add uniqueness. Examples will follow throughout this document.

In many cases, it is desirable to derive a new ID from an existing EIDR ID rather than obtain a unique EIDR ID for an object. ALID, identifying an entitlement in an Avail, is a good example. In these cases, it is strongly preferred to indicate that the identifier is not being used in the correct context by using EIDR-X rather than EIDR-S. That is, use the eidr-x ID scheme, and construct the ID using the EIDR-X format.

3.3 EIDR Object Type

When using EIDR ~~as the based for an AvailID~~ it is important to use the correct EIDR Object Type. EIDR Object Types are defined in [EIDR-UG], Section 4.

EIDR Types should be assigned based on the asset types ~~in the Avail.~~ In general if there is a single asset (or season) the EIDR type should represent a particular Edit of that asset. If there is a collection of objects, it should be an EIDR Compilation (collection of assets). However, if this is impractical, the type of the primary asset can be used.

Asset Type	EIDR Type	Alternate EIDR Type
Movie	Edit	
Episode	Edit of Episode	
Season	Season	
Movie with extras	Compilation	Edit
Episode with extras	Compilation	Edit
Season with Extras	Compilation	Season

3.3.1 EIDR in Avails

In general, the EIDR-X form is most appropriate for ~~Avails~~ALID. This is because there may be multiple Avails for a single title. The <extension> part makes the Avail unique. For example,

let's assume a single title: The Devil is in the Details, EIDR Edit = 1012-7947-21D5-9D24-CC5F-H. ~~AvailIDs~~ALID might be:

```
md:availalid:eidr-x:1012-7947-21D5-9D24-CC5F-
H:craigsmovies.com_july_NorthAmerica
md:availalid:eidr-x:1012-7947-21D5-9D24-CC5F-H:craigsmovies.com_july_Europe
md:availalid:eidr-x:1012-7947-21D5-9D24-CC5F-H:craigsmovies.com_aug_NorthAmerica
md:availalid:eidr-x:1012-7947-21D5-9D24-CC5F-H:craigsmovies.com_aug_Europe
```

Because the extended EIDR (EIDR-X) form was used, it is possible to distinguish retailers, timing and region. This is done in a human-readable form, but it could also be done with something computer-friendly.

Naming conventions in extensions are intended to support uniqueness human readability and SHOULD NOT be parsed automatically to extract information.

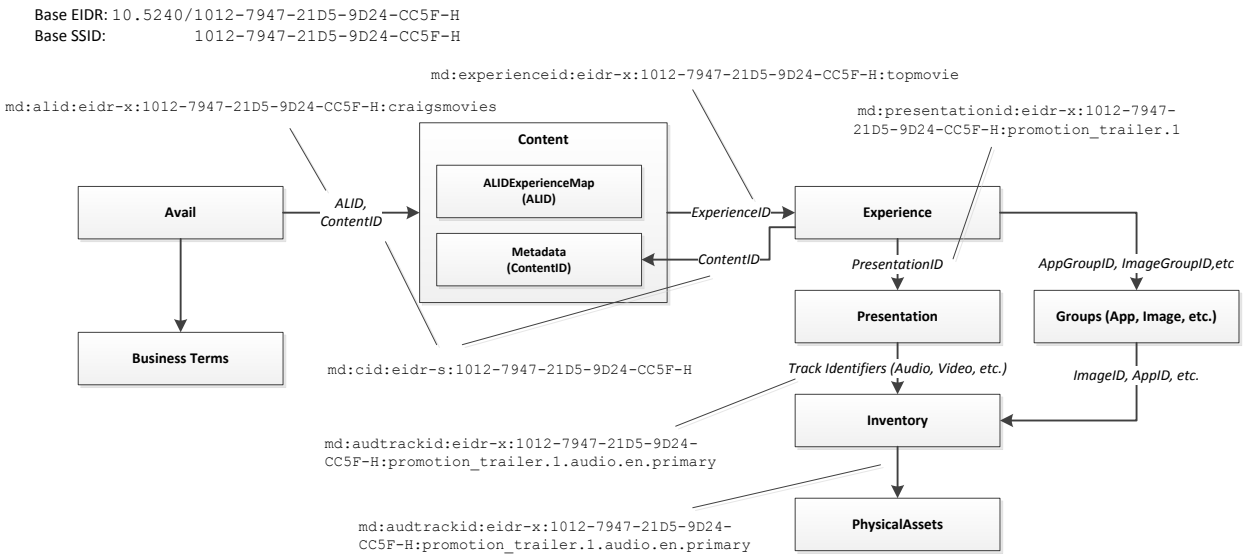
To simplify processing, Avails provides the means to provide multiple levels of EIDR in the metadata. For example, Asset/Metadata includes TitleEIDR-S and EditEIDR-S. Analogies are in EpisodeMetadata, SeasonMetadata and SeriesMetadata.

3.4 Practice for Constructing Asset Identifiers from top-level IDs

This section defines a recommended practice for building the IDs required by the Manifest. This practice is designed to make the manifest easier to create, read and maintain.

This practice should be used when unique identifiers do not already exist for an object. For example, if there is an ID for a movie's trailer, it should be used. However, it there is no ID for that trailer, this practice describes how to construct a unique trailer ID from the movie's ID.

Following is an example of identifiers using this convention.



ID form remains consistent with the other ID recommendations. That is, the ID is of the form:

<MDID> ::= "md:"<type> ":"<scheme>":"<SSID>

This practice addresses the <SSID> part. The following form is used. Note: It looks complicated, but you'll see from the examples it's quite simple.

<SSID>::= <Experience SSID> | <Presentation SSID> | <Component SSID>

When constructing an identifier, it is important that the completed <SSID> is formatted in compliance with the <scheme>.

The following examples use EIDR IDs; for example, `md:presentationid:eidr-x:1012-7947-21D5-9D24-CC5F-H:trailer.1`. Other identifier schemes also work, provided one can distinguish between the pure identifier and one with an extension. This practice suggest appending '`-x`' to the org-specific scheme. For example, `org:craigsmovies.com` becomes `org:craigsmovies.com-x` as shown in this example, `md:presentationid:org:craigsmovies.com-x:abc123:trailer.1`

3.4.1 Derived Experience IDs

Experience IDs are derived using the <Experience SSID> form.

<Experience SSID> ::= <Root SSID>[":"<experience name>]

- <Root SSID> refers to the identifier of the parent object from which the Experience is derived. For example, if the Experience is for a movie, the <Root SSID> is the SSID for the movie. If the Experience is for a season, the <Root SSID> is the SSID for the season.
- <experience name> is a string unique for Experience IDs for the <Root SSID>. That is, it needs to be different for each ExperienceID derived from a given <Root SSID>.

As Experiences often align with the top-level identifier used in ALID and ContentID (i.e., <Root SSID>. In this case, the Experience ID is simply constructed using the base identifier as the SSID; for example, `md:experienceid:eidr-s:1012-7947-21D5-9D24-CC5F-H`.

Where there is not a simple mapping Experience ID can be constructed by appending unique data; for example, `md:experienceid:eidr-x:1012-7947-21D5-9D24-CC5F-H:exp1`.

3.4.2 Derived Presentation IDs

Presentation IDs are derived using the <Presentation SSID> form.

<Presentation SSID> ::= <Presentation Unique ID> | <Root SSID>":"<av type> ["_"<av subtype>] ["."<index>]

- <Presentation Unique ID> and <Component Unique ID> are identifiers that are assigned to the assets externally to this practice. For example, if an EIDR ID is

created for a particular Presentation that would be the <Presentation Unique ID>. Similarly, if an EIDR ID is created for a given audio track, that would be the <Component Unique ID>.

- <Root SSID> refers to the identifier of the parent object from which the Presentation is derived. For example, if the Presentation is a trailer for a movie, the <Root SSID> is the SSID for the movie. It is assumed there is only one "feature" presentation for the given <Root SSID>
- <av type>, <av subtype> and <index> distinguish this asset from other assets
  - <av type> is as defined in Audiovisual/Type.
  - <av subtype> is as defined in Audiovisual/Subtype. This should be included if Audiovisual/Subtype is present.
  - <index> is a number used to differentiate objects of the same type/subtype.
  - For example: `main, promotion_trailer.1, trailer.2, bonus_making-of.1, bonus_making-of.2, bonus_deleted-scenes.1, bonus_deleted-scenes.2`

Following are examples of Presentation IDs using this approach:

- `md:presentationid:eidr-x:1012-7947-21D5-9D24-CC5F-H:main`
- `md:presentationid:eidr-x:1012-7947-21D5-9D24-CC5F-H:trailer.1`
- `md:presentationid:eidr-x:1012-7947-21D5-9D24-CC5F-H:bonus_deleted-scenes.1`

3.4.3 Derived Component IDs

Componet IDs are derived using the <Component SSID> form.

<Component SSID> ::= <Component Unique ID> | <Presentation SSID> "."<component type> ["."<index>]

<component type> ::= "video" | <audio type> | <subtitle type>

<audio type> ::= "audio."<language>["."<audio type>"]

<subtitle type> ::= "subtitle."<language>["."<subtitle type>"]

- <language> is a language code as defined in [CM]
- <audio type> is as defined in Inventory/Audio/Type. Additional information can be appended as necessary for clarity or uniqueness.
- <subtitle type> is as defined in Inventory/Subtitle/Type. Multiple types or additional information can be appended as necessary for clarity or uniqueness.

- For example, `video`, `audio.en.primary`, `audio.fr.narration`, `audio.es.dialogcentric`, `audio.de.commentary`, `subtitle.fr.normal`, `subtitle.en.forced subtitle.en.sdh`, `subtitle.de.large`, `subtitle.es.commentary`

Following are examples of component IDs using this approach:

- `md:vidtrackid:eidr-x:1012-7947-21D5-9D24-CC5F-H:main.video`
- `md:audtrackid:eidr-x:1012-7947-21D5-9D24-CC5F-H:main.audio.de.commentary`
- `md:subtrackid:eidr-x:1012-7947-21D5-9D24-CC5F-H:bonus_trailer.1.subtitle.de.large`

Physical identifiers such as found in Inventory/{Audio|Video|Subtitle}/TrackIdentifier can be represented as follows. Note that physical assets are generally identified as an APID (Physical Asset ID).

- `md:apid:eidr-x:1012-7947-21D5-9D24-CC5F-H:main.video`
- `md:apid:eidr-x:1012-7947-21D5-9D24-CC5F-H:main.audio.de.commentary`
- `md:apid:eidr-x:1012-7947-21D5-9D24-CC5F-H:bonus_trailer.subtitle.de.large`

3.5 IDs, Computer-readability and Human-readability

IDs are designed to be computer readable. The most important factors are that the ID be well formed and unique with the scope of usage.

Some conventions such as ID type are structurally important as they allow unique identifiers to be assigned using the same root identifier (e.g., EIDR ID).

Where possible, we used conventions that would enhance the readability of the ID. For example, you can distinguish a Content ID from an Experience ID by the type embedded in the string. You can determine an ID type by viewing the scheme.

NEVER USE STRINGS FROM AN IDENTIFIER IN LIEU OF METADATA. For example, you might see an ID that looks like this: `md:subtrack:eidr-x:1012-7947-21D5-9D24-CC5F-H:subtitle.de.large`. We certainly hope this is a German large subtitle, but it might not be. Rely on the appropriate metadata (e.g., Track/Subtitle/Language). The only reliable aspect of any identifier is that it is unique.

3.6 ID Summary

Unless otherwise noted:

- ~~AvailID is~~ALID must be unique for the Avail~~. It~~ for a given Licensor and should be ~~built from the base EIDR ID in EIDR x form.~~globally unique.

- ALID generally refers to an Edit of the Movie, TV episode, etc.; or a compilation of various assets. It should be built from the base EIDR ID in EIDR-s or EIDR-x that most closely matches the related assets.
- Asset/@contentID – references metadata for the movie. It should in EIDR-s form. This is informative only as metadata might not yet be finalized.

Avails are connected to Experiences as follows:

ID	Avails Type	Mapping to Experience
Avail IDALID	Avail/@AvailLID	MediaManifest/AvailsExperienceMap
ALID	Avail/ALID	
Content ID	Avail/Asset/@contentID	Experience/Audiovisual/@ContentID

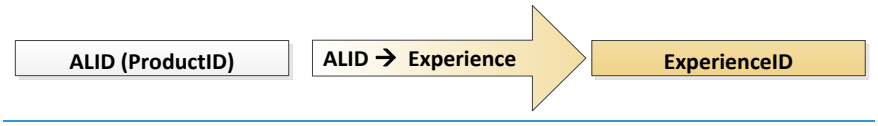
4 CONNECTING OBJECTS

This section describes how one gets from Avail to Media Manifest and from Media Manifest to physical (encoded) assets.

The mappings can be simple and direct. However, in practice information is often incomplete or inconsistent. This section attempts to address the real-world scenarios to allow mapping under various non-ideal conditions.

4.1 Mapping Avail to Experience

The most direct means to map Avail to Experience is to use the Avail Experience mapping found in the Media Manifest (MediaManifest/A~~vail~~LIDExperienceMap). This defines the set of Experiences that satisfy a set of Avails.



Selection of the precise Experience then involves looking for the Experience that matches language and region~~.~~ as specified in the Avail. Information on region can be found in Section 8.1. In XML, the ALID is in Avail/ALID. In Excel, the 'ALID' is determined as follows:

Type of Avail	Excel, when using EIDR IDs	Excel, if using custom ID equivalent
Movie/Episode	ProductID	AltID
Season	SeasonContentID	SeasonAltID
Series	SeriesContentID	SeriesAltID

If, for some reason, the three elements are not simultaneously available it is still possible to infer which Experience applies to which Avail. However, this is not a general solution and might not work in all circumstances. The Retailer and the Distribution Entity organization must agree upon constraints that allow this work.

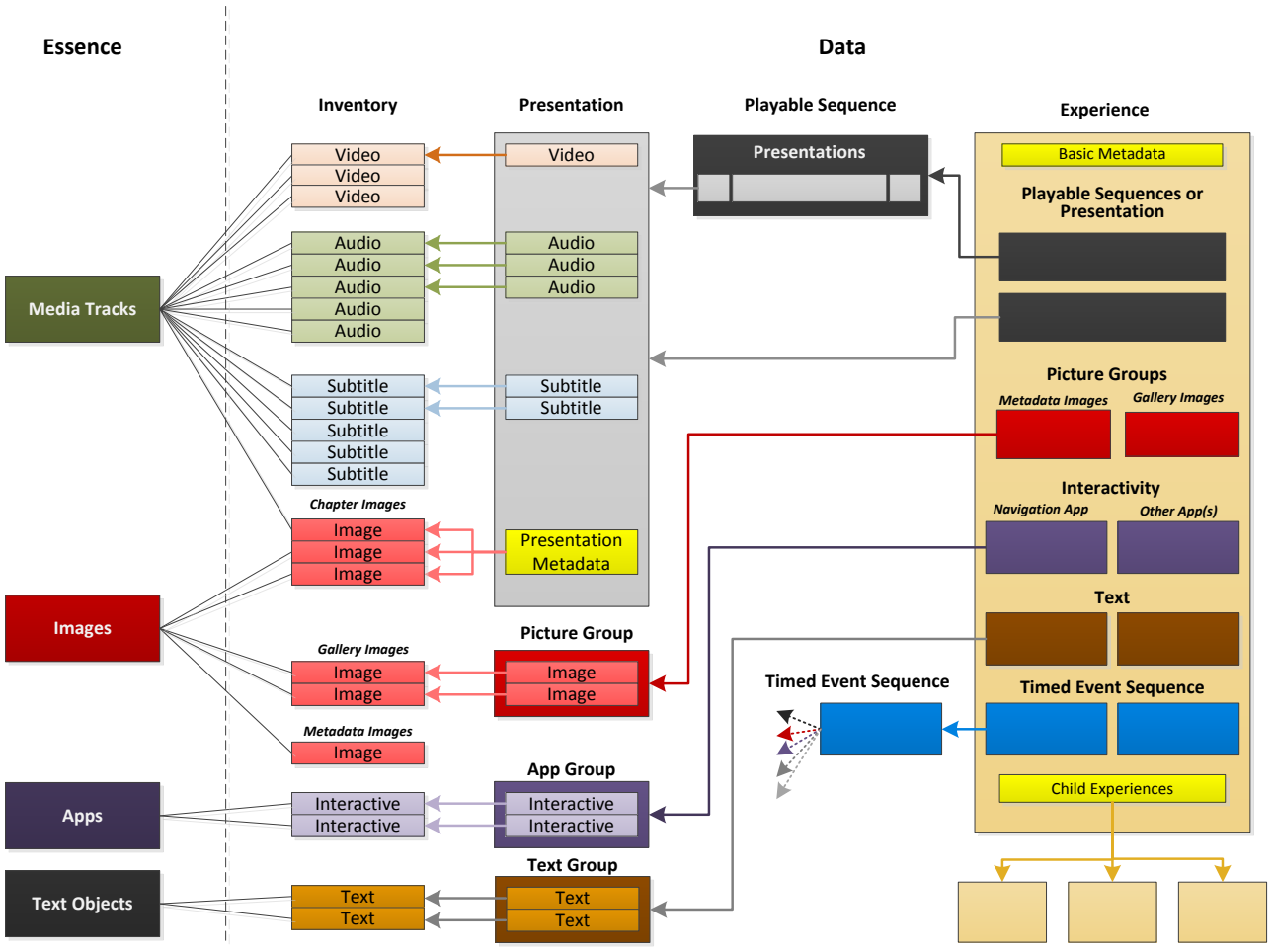
Following are some examples of what could work *under controlled circumstance*:

- Assuming only a single asset and one release per region, Experience could be matched based on Avail/@ContentID matching Experience/BasicMetadata/@ContentID, and region/territory matching as per Section 8.1. One could, by convention, use Avail/Asset/Metadata/ProductID to match Experience/BasicMetadata/@ContentID.
- Assuming multiple assets and one release per region, Experience can be matched based on ProductID as in the previous example. One would have to identify an Experience that contained all the ProductIDs from all the Assets in the Avail. This would require constraints on the Avail to avoid assets mapping to more than one Experience.

4.2 Mapping Avail file references to Media Manifest asset references

The Media Manifest provides a very direct means to map Experience to Assets.

In the following diagram, the vertical dotted line shows where this mapping occurs. This process is described in detail in [Manifest], Section 2.2.3.



Media Manifest provides the means to reference content either by asset identifier (physical asset) or by location.

4.2.1 Referencing assets by identifier

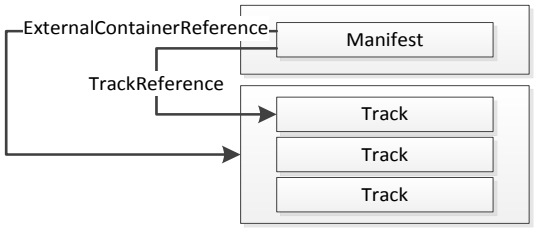
The most general and correct is to use TrackIdentifier in various Inventory types (Audio, Video, etc.). This allows assets to be referenced regardless of how the asset is containerized or packaged with other tracks. However, the Retailer's asset management system must be able to index off identifiers. Retailers might wish to consider this on their roadmaps.

4.2.2 Referencing assets by location

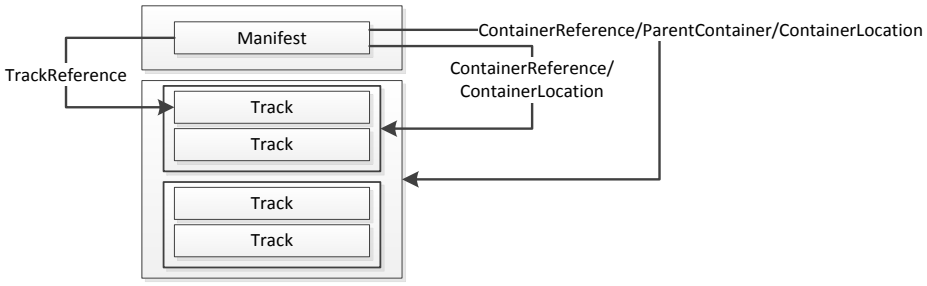
In this recommendation, the only location we are considering is filename. It is presumed that any other types of location (e.g., URL for download) will be resolved through the File Manifest that enumerates delivery mechanisms.

The element for referencing filenames is ContainerReference which is found under each of the Inventory media types (Audio, video, etc.). The outermost container in Inventory's ContainerReference/ContainerLocation/… should match the File Manifest's FileInfo/Location.

Consider the following diagram from [Manifest]. File Manifest FileInfo/Location corresponds with the Media Manifest's ExternalContainerReference.



The following illustrates track within nested containers. In this scenario, File Manifest's FileInfo/Location corresponds with the *outermost* container, in this case ContainerReference/ParentContainer/ContainerLocation. Note that the File Manifest does not address the internal structure of a container—the only possible reference is the file itself which is the outermost container.



4.2.3 Images

There are several types of images in a Media Manifest; for example, chapter images, metadata images and gallery images. It is important the recipient of a Media Manifest be able to easily locate images associated with an Experience.

All Images must be included in the Inventory.

Images that are part of an Experience include the following:

- Images in Picture Group referenced in Gallery/PictureGroupID

- Images referenced in BasicMetadata/LocalizedInfo/ArtReference. Basic Metadata exists in Experience and Audiovisual, both through reference (ContentID) and by inclusion (BasicMetadata).
- Images referenced in Presentation/Chapters/Chapter/ImageID for all Presentations referenced directly by the Experience, or referenced indirectly through a Playable Sequence

To make it easier to locate images associated with an Experience, the recommended practice is to also include images that belong together (e.g., chapter images and metadata images) in a Picture Group. With the exception of Picture Groups referenced in Galleries, Picture Groups associated with an Experience should be referenced in Experience/PictureGroupID. Specifically

- All chapter images for a Presentation should be in a Picture Group
- All metadata images associated with a ContentID should be in Picture Group

References to images in BasicMetadata/LocalizedInfo/ArtReference should be to the Inventory. That is, the reference should be of form "md:imageid:"<scheme>":"<SSID>. For example:

```
<manifest:Audiovisual ContentID="urn:dece:cid:eidr-s:C5CA-AD07-3B62-A2B0-23C1-G">
 <manifest:PresentationID>presentation</manifest:PresentationID>
 <manifest:BasicMetadata ContentID="urn:dece:cid:eidr-s:C5CA-AD07-3B62-A2B0-23C1-G">
   <md:LocalizedInfo language="en" default="true">
     ...
     <md:ArtReference resolution="800x600">md:imageid:eidr-x:C5CA-AD07-3B62-A2B0-23C1-
G:poster.en</md:ArtReference>
     ...
   </md:LocalizedInfo>
   <md:LocalizedInfo language="de">
     ...
     <md:ArtReference resolution="800x600"> md:imageid:eidr-x:C5CA-AD07-3B62-A2B0-23C1-
G:poster.de</md:ArtReference>
     ...
   </md:LocalizedInfo>
   ...
 </manifest:BasicMetadata>
</manifest:Audiovisual>
```

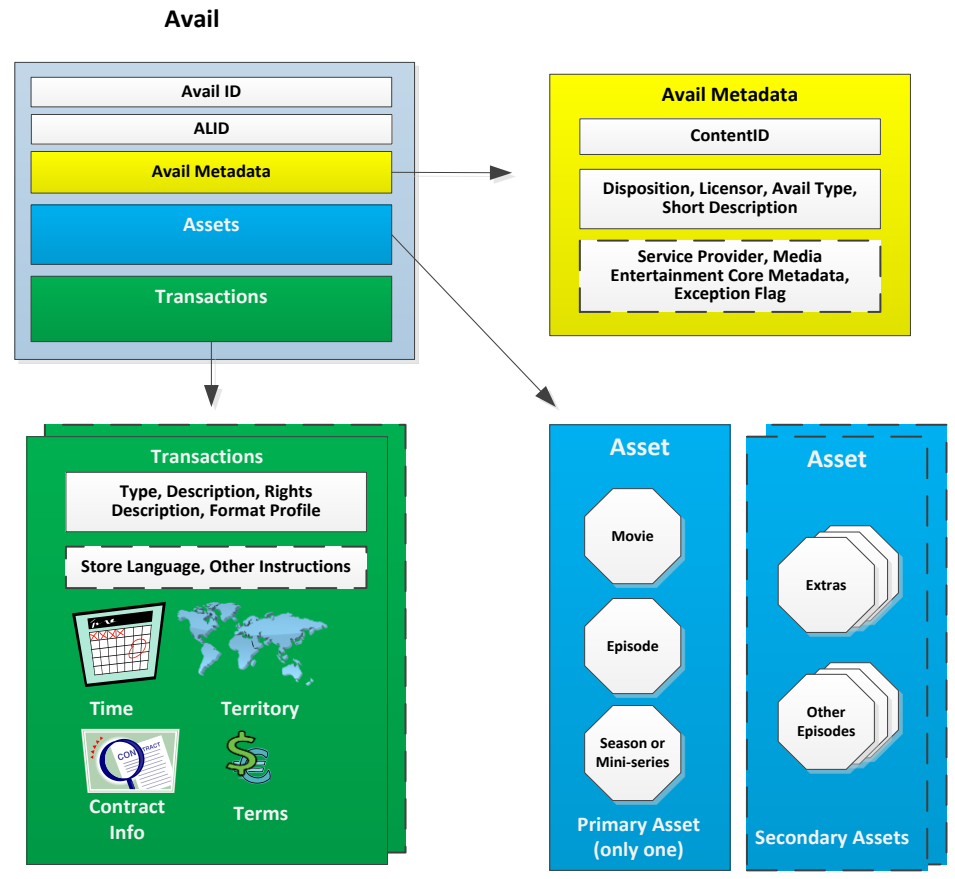
5 AVAILS

An Avail is one or more assets and one or more business rules that define when, where and for how much those assets are offered and delivered to a user.

An example of Avails can be found in [Avails]. Although this model is generally compatible with any avails description mechanism, this document will use [Avails] as the model for the purposes of description.

5.1 Avails Model

A generalized representation of Avails is shown in the following figure:



On the left is a generalization of the Avails structure. On the right is a conceptualized expanded view of Assets and Transactions, more precisely Avail/Asset and Avail/Transaction respectively. AvailMetadata covers other Avail data such as disposition, licensor, service provider, avail type, description and notice that human intervention is required. These other data are described in [Avails] and are not directly relevant to this discussion.

What is important for this discussion is how Avails relates to the assets that are delivered to the Retailer and ultimately to the consumer.

5.2 Constructing ~~an AvailID~~ALID

~~An AvailID~~The combination of ALID and Licensor SHALL be globally unique. ~~The same AvailID~~ALID SHOULD be globally unique. A Licensor SHALL NOT ~~be used~~use an ALID for a different combination of assets and terms. An Avail associated with an A~~vail~~LID SHALL have only one disposition.

An A~~vail~~LID SHOULD comply with the A~~vail~~LID format above. This is not a strict requirement, but it will make global uniqueness much easier and avoid us IDs in the wrong context. Note that UUIDs avoid the first issue, but not the second.

An A~~vail~~LID SHOULD be based on an EIDR ID.

An ALID defines the product. See Section 4.1 for information on deriving ALID from Excel Avails. Accordingly, the same ALID may be used in more than one Avail.

When an ALID refers to a single asset (e.g., a movie, a TV season or a TV episode), it should contain the same EIDR ID as the asset's ContentID.

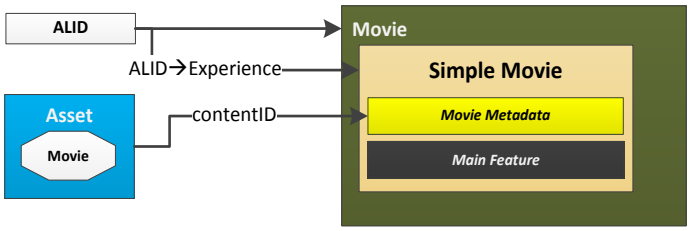
5.3 Avails of various Experiences

The following illustrations show how Avails and Experiences would be constructed around a few types of titles.

The most common cases are a single move, a single episode and a single season. These are described in Sections 5.3.1 and 5.3.2. Other cases are described in the sections that follow.

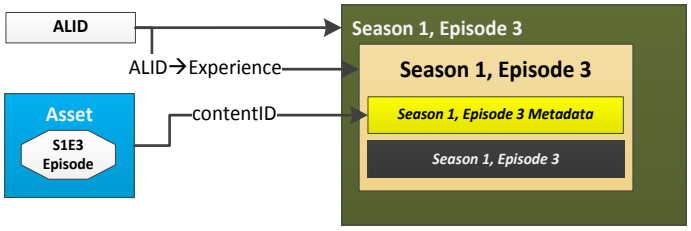
5.3.1 Availing a single Movie or TV Episode

The Avail for a Movie has a single Avail/Asset element describing the Movie.



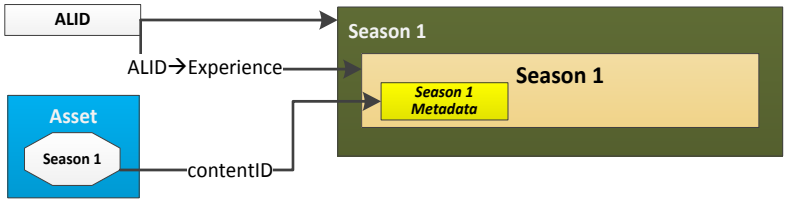
A single entry is in ALIDExperienceMap mapping the ALID to the ExperienceID for the movie.

The structure is the same for a single Episode. The following example is for a single episode, in this case Season 1, Episode 3. In general, each episode would get its own Avail and Experience.



5.3.2 Television Season without episodes listed

The season without episodes explicitly listed as distinct Assets is similar to the Season without episodes listed as Assets (later example). However, since there is only one Asset (the season itself) and episodes might not yet be known, it is does not make sense to reference individual episodes. It is understood from context that the Experience will ultimately include episodes. This model is used for presale ("Season Pass") or a complete season.



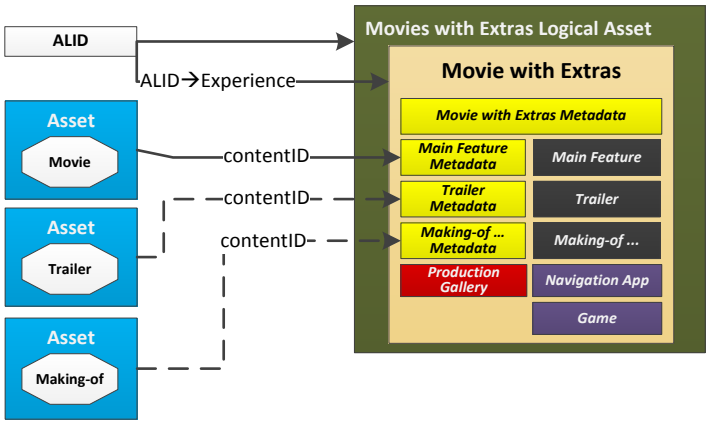
In this case, the ALID Asset/@contentID both are built from the season-level identifier.

5.3.3 Movie with Extras ~~and Season with episodes listed~~

The following illustrates a movie with bonus material (extras~~) and a season with selected episodes. In both cases the~~). The ALID represents an object that includes all assets (e.g., EIDR Compilation).

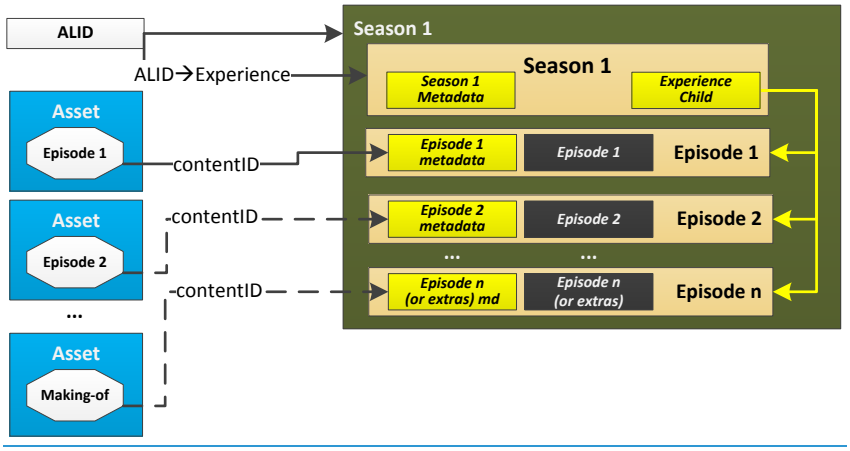
~~In both cases,~~ Asset elements are created for each Asset. In each of these Assets, metadata describes that individual Asset and the ContentID refers to metadata for that Asset. Note that these identifiers are for the Asset, not for the season/collection. Typically, these will be the EIDR Edit for that work.

The following movie example allows specific assets to be listed. This avoids any ambiguity about which assets are included.



5.3.4 Season by Episodes

The following form is used when it is important to be explicit about the episodes that are included. In the previous example, the entire season is included. In the following example, certain episodes can be excluded. Generally, this would not be used for complete seasons. Note that there is an Experience for the season and also one for each episode.



~~5.3.4~~5.3.5 Avail with Multiple Experiences

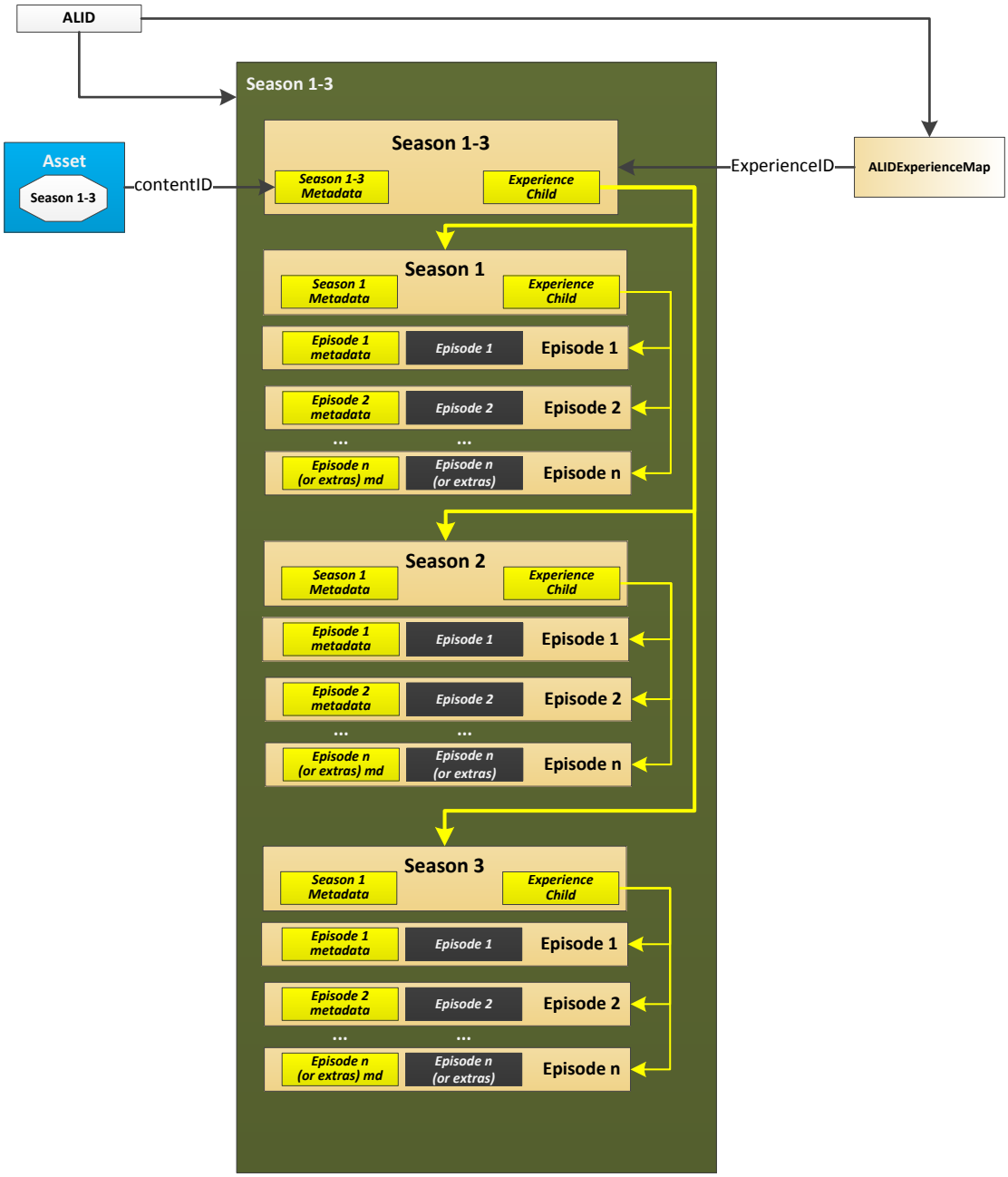
These examples illustrate how an ALID can identify a set of Experience elements. In these examples, there is an Avail that covers three seasons—although it could be any combination of content. An ALID is defined that covers those three seasons.

In both cases, an ALID is created that covers three seasons. This is independent of how the Avail metadata is constructed or how the Experiences are structured.

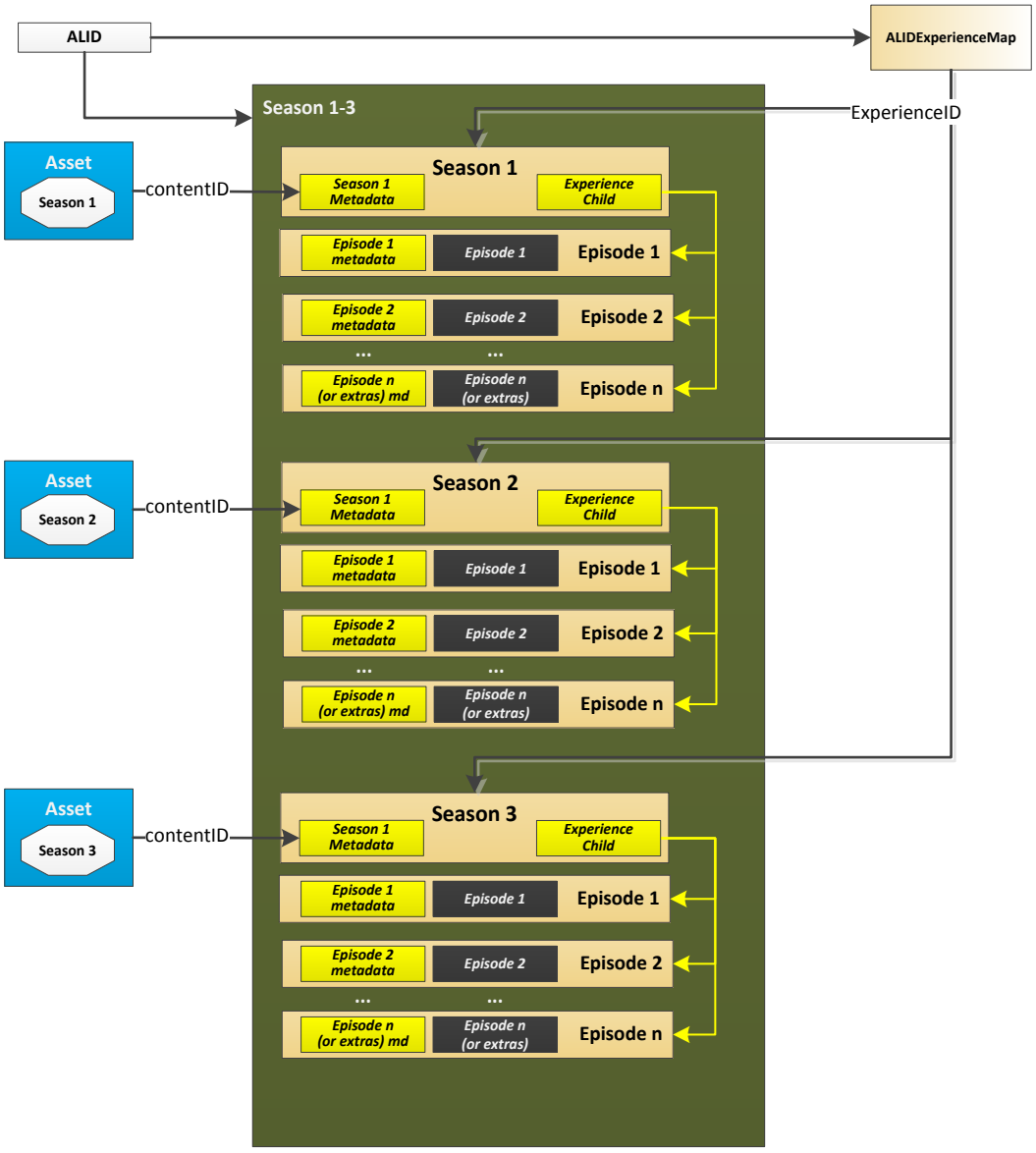
In the first case, the three seasons are presented to the consumer as a single entity, including its own metadata. For example, these seasons might be sold as "Show XYZ, *The Early Years*". Consequently, it is viewed as a single entity both in the Avails Asset as well as from the Experience. The Avail is created with one Asset. Subsequently, an Experience will be created for the three

seasons. Each individual season still exists as its own experience, but the parent Experience (three seasons) links to those through ExperienceChild.

The ALID need only map to the single top-level Experience as the others are inferred through the linkages.

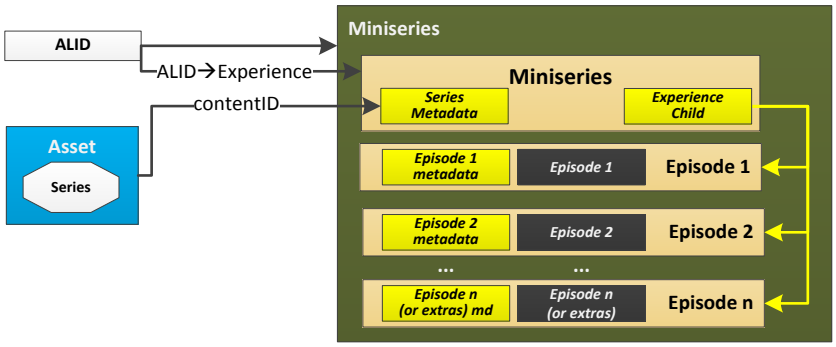


In the second example, the Avail lists each season as its own Asset. While the previous example implies that the set of seasons is a distinct entity, this example makes no attempt to group the seasons other than to say they're sold together. There is an Avail/Asset element created for each season. ALIDExperienceMap maps the single ALID to the collection of Experiences.

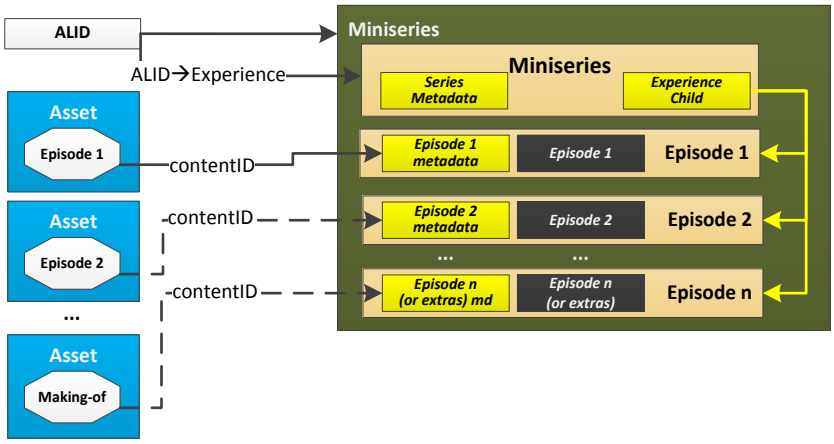


5.3.6 Availing Miniseries

Miniseries are availed like seasons with a few exceptions. Within /Avail/Asset, SeriesMetadata is used in lieu of SeasonMetadata. Miniseries are organized like Series, but child references are to episodes rather than seasons. This is illustrated in the following two figures. In the first, the miniseries is availed (typical).



In the second, individual episodes are availed. Note that in the Avail, AssetMetadata/EpisodeMetadata uses SeriesMetadata instead of SeasonEpisode that would be seen for multi-season episodic TV series.



For information on organizing the Experience, see Section 6.5.1.3.

5.4 Holdbacks

A holdback is a special type of Avail that indicates exceptions to another Avail.

5.4.1 Holdback terms

Holdbacks are defined in Terms with the termName values 'HoldbackScope', 'HoldbackAsset', 'HoldbackAssetType', and 'HoldbackLanguage'. A Holdback should not have any other terms.

Holdback territories, time periods and assets must be a subset of the original Avail. Note that by subset, it could match exactly or represent a smaller set (proper subset).

5.4.2 Holdback examples

Let's say a studio had licensed a movie worldwide for EST, but wanted to withhold the German Subtitle track for weeks 2 and 3. There would be two Transaction elements. The first licensed the movie, and would have the appropriate terms.

The second, the holdback avail, would look exactly the same, except it only covered weeks 2 and 3.

This holdback avail could hold back specific tracks as follows:

```
<avails:Term termName="HoldbackAsset">
    <avails:URI>urn:dece:subtrackid:eidr-x:C5CA-AD07-3B62-A2B0-23C1-G:main.subtitle.de</avails:URI>
</avails:Term>
<avails:Term termName="HoldbackAsset">
    <avails:URI>urn:dece:subtrackid:eidr-x:C5CA-AD07-3B62-A2B0-23C1-G:commentary.subtitle.de</avails:URI>
</avails:Term>
```

Or, the holdback avail could hold back by language and track type

```
<avails:Term termName="HoldbackAssetType">
            <avails:Text>Subtitle</avails:Text>
</avails:Term>
<avails:Term termName="HoldbackLanguage">
            <avails:Text>de</avails:Text>
</avails:Term>
```

If the holdback prohibited download and license (i.e., sale, rental and streaming allowed), the following terms could be added:

```
<avails:Term termName="HoldbackScope">
            <avails:Text>Download</avails:Money>
</avails:Term>
<avails:Term termName="HoldbackScope">
            <avails:Text>License</avails:Text>
</avails:Term>
```

5.5 Determining Which Tracks Are Included in an Entitlement

A player only plays what a user has acquired. This section describes the process for mapping the acquisition to specific tracks. The goal is to ensure that a user gets the Experience associated with the entitlement (e.g., movie-only vs. movie with bonus features) and plays the tracks associated with the entitlement (e.g., 'SD' vs 'HD', and director's commentary or not)

A player must determine whether individual tracks fall into the scope of an Avail. This is done by matching the disposition of an entitlement (e.g., the user purchased the title in 'HD') and information in the Avail for that entitlement with information in the Manifest.

This scenario assumes the user has acquired content in accordance with a particular Avail.

Selection is based on the following information from the Avail against which the content was acquired. This information is as follows:

- ALID – The ALID associated with the entitlement. ALID is in the Avail/ALID element.

- Condition – The current status of the entitlement. Condition is in the Transaction/ExperienceCondition element. Standard condition values are documented in [Manifest], Section 9.2. However, other conditions could exist.
- Media Profile (Format Profile) – Characteristics of the media in general terms (e.g., 'HD' and 'SD'). Profile terms are document in [Avails], Section 2.2.3

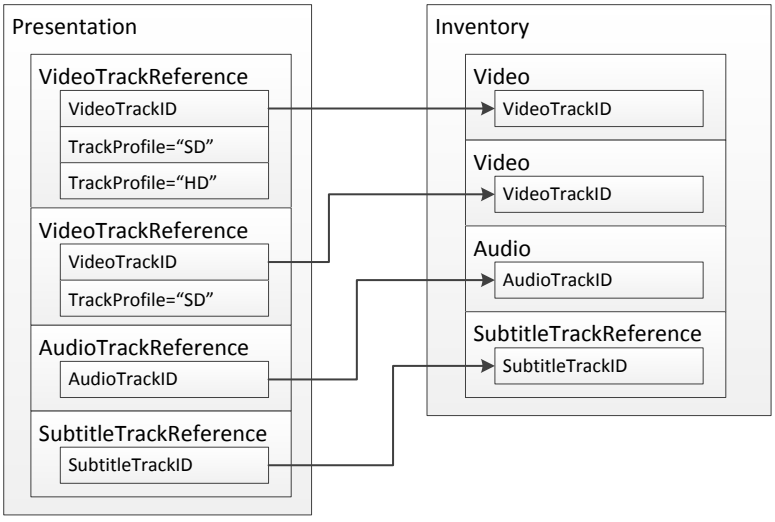
Given that information, and a properly constructed Media Manifest, the Player can determine which Experience to use and which tracks are covered by the entitlement.

First the player selects the correct Experience. ALIDExperienceMap allows ALID and Condition to map to a set of Experiences. That is an Experience maps if Avail/ALID from the Avail matches ALIDExperienceMap/ALID in the Manifest and Avail/Transaction/ExperienceCondtion matches ALIDExperienceMap/ExperienceID/@condition for the Experience's ID.

This set of Experiences is then downselected based on player settings such as Region and Language.

The Experience references one or more Presentation, possibly indirectly through Playable Sequence. Within that Presentation, a track is playable if its Profile matches the Avail. That is, the Avail's Transaction/FormatProfile matches any instance of the track's TrackProfile (TrackMetadata/AudioTrackReference/TrackProfile, …/VideoTrackReference/TrackProfile, etc.). Note that a Track that does not have any TrackProfile instance is assumed to match all profiles.

The following illustration includes a Presentation that includes both an HD and SD tracks, the HD track will have TrackProfile='HD' and TrackProfile='SD'; and the SD track will have TrackProfile='SD'. If FormatProfile='HD', both tracks are matched. If FormatProfile='SD', only the SD track is matched. There is also an audio track and a subtitle track with no FormatProfile. The audio and subtitle tracks match all profiles.



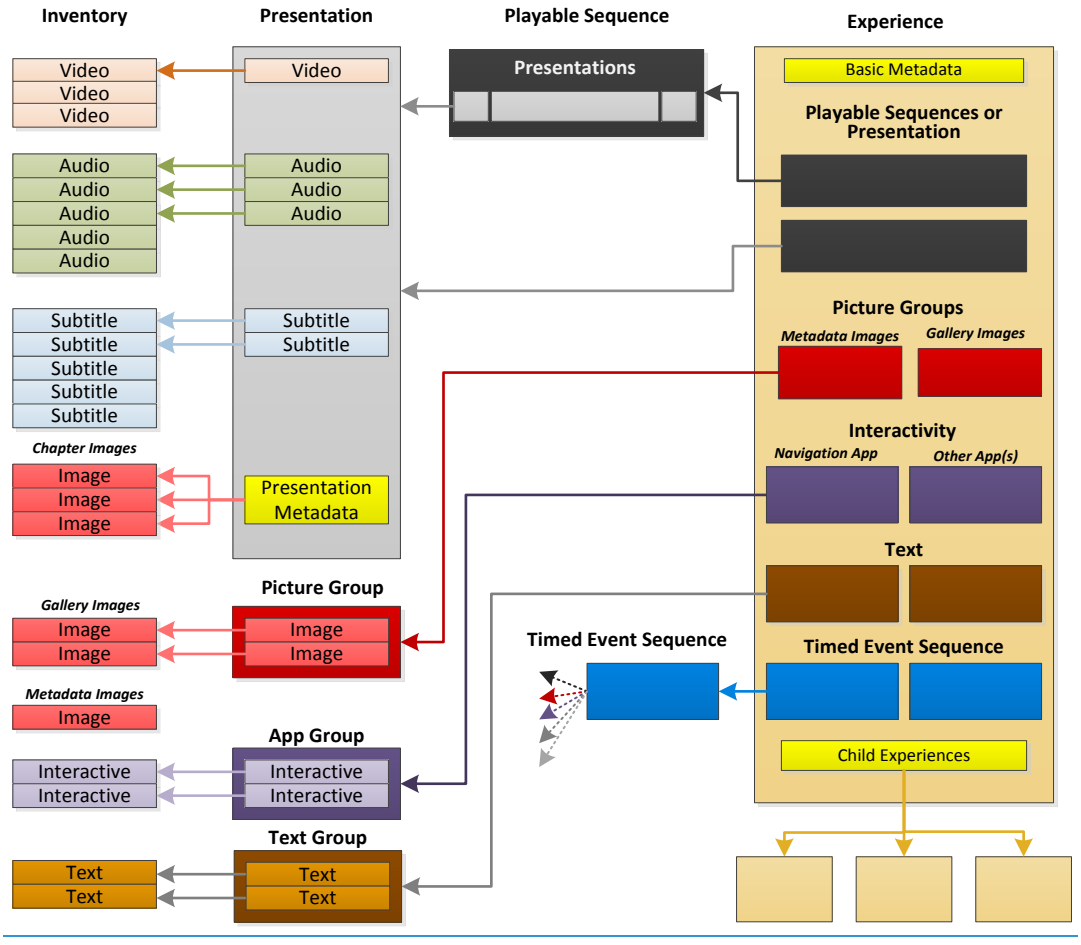
6 MANIFEST CONSTRUCTION

This section provides additional guidance on Media Manifest construction.

6.1 References within Manifest

The following diagram represents a Media Manifest and encoded media associated with the Manifest. This is described in detail in [Manifest].

Generally, arrows represent references identifier, although in at least one case (Playable Sequence) it represents optional inclusion.



How identifiers are used for references are determined by whether the references are to be used entirely within the Media Manifest or whether those identifiers will be external references. If references are internal, identifiers can be in any form as long as they are unique.

External identifiers should be used when they are available. For example, a Presentation might correspond with a Presentation in a Common Media Package (CMP). Use of the same Presentation ID allows the CMP to be used for media playback in lieu of the Presentation structure in the Manifest (or vice versa).

It is easy to confuse identifiers, so we recommend using the naming conventions in Section 3.1.7**Error! Reference source not found.**.

We offer no recommendation on whether Playable Sequence should be included or referenced in Audiovisual.

6.2 Region and Language

Region and language should be used as necessary to provide the correct experience to the user.

Region should be encoded using guidance from Section 8.1. The primary reasons for using region are to provide distinct playable sequence and to offer different sets of tracks to different regions. If one is not doing one of those two things, they should carefully consider whether regions are required. Note that Avails should prevent the availability of content to given territories. Many consider it a feature to offer the broadest set of languages to their customers.

Languages can, in general, be handled through track selection, although for marketing reasons, some may opt to group tracks into different experiences. Note that metadata localization allows one metadata set to be internationalized so it can be used across multiple languages and regions.

6.3 Metadata

The within the Experience is means to convey Metadata describing an Experience, Playable Sequence or Presentation. There is metadata at the experience level, and also at the Audiovisual.

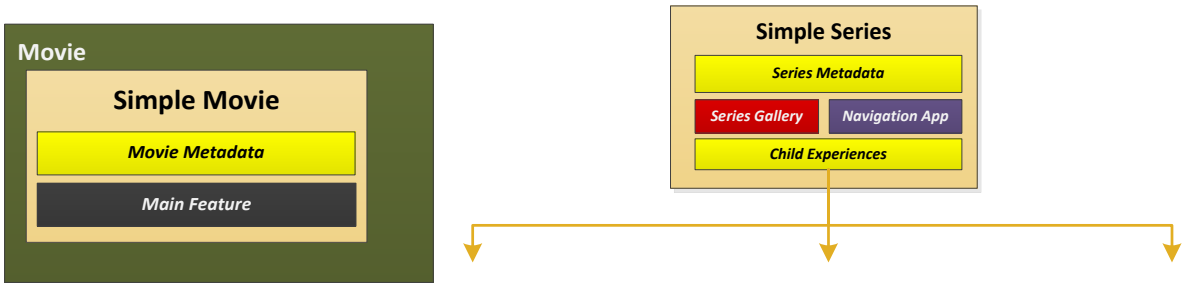
6.3.1 BasicMetadata by reference or inclusions

The implementer is given the option of including metadata in the XML document, or referring to the metadata elsewhere. This is reflected by an xs:choice between a BasicMetadata element and a ContentID element. When using BasicMetadata, metadata is included. ContentID refers to metadata with the assumption the recipient can locate that metadata.

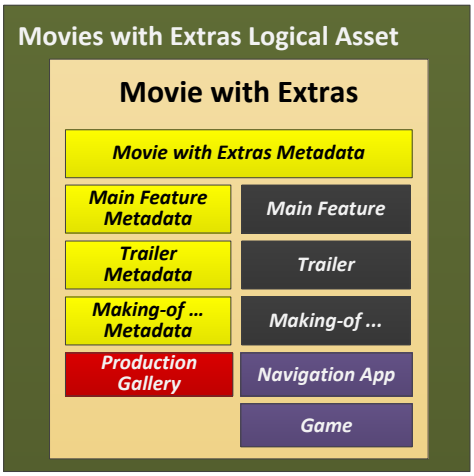
We strongly recommend metadata be included by reference (ContentID) rather than included in place (e.g., Experience/ContentID rather than Experience/BasicMetadata/…). We found that metadata in place makes it more difficult to maintain both the Experience and the metadata. With Manifest version 1.3, metadata was added to the Inventory.

6.3.2 Metadata requirements for Experience and Audiovisual

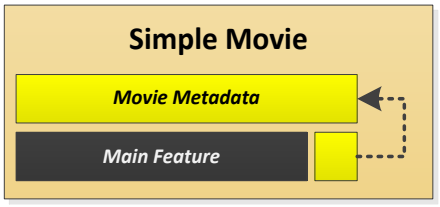
There must always be metadata for the Experience. This provides any application that needs information about the Experience to obtain that information and present it to a user. This applies whether the Experience is a single asset or is part of a complex hierarchy as illustrated below:



Metadata is also needed to describe each audiovisual asset within an Experience. The following example shows a movie with additional materials. There is metadata at the Experience level and also metadata for each audiovisual asset.



In the special case when the Experience metadata matches the Audiovisual metadata, the Audiovisual metadata should be by reference. That is, use Audiovisual/ContentID using a Content Identifier that matches Experience/ContentID or Experience/BasicMetadata/@ContentID.



6.3.3 Metadata in Inventory

As noted in Section 6.3.1 it is preferred that metadata be included as its own object rather than in embedded in other elements (e.g., Experience/ContentID rather than Experience/BasicMetadata/…). As of Manifest version 1.3, Metadata can be included in the Inventory. This is described in [Manifest], Section 4.2.7.

The best usage for delivery is to include metadata in Inventory/Metadata/BasicMetadata. The ability to reference files is supported, however, this usage should only be supported if there is a specific need to deliver metadata out of band.

In cases where a BasicMetadata object will be used with different localizations, the Alias mechanism as described in [Manifest], Section 4.2.7.1 can be used. Applications ingesting metadata should be prepared to match both Metadata/@ContentID and Metadata/Alias/@ContentID.

6.3.4 Required Metadata

6.3.4.1 Episodic

When the metadata describes an episode, BasicMetadata has the following constraints

- ReleaseDate is required to at least date resolution.
- Sequence need not be included. It is redundant and potentially contradictory with ExperienceChild/SequenceInfo. See Section 6.5.1.
- When an episode has 'episode thumbnail' LocalizedInfo/ArtReference is to be used
- When ratings are available for an episode, it must be included in the Ratings element.
- When ratings are available the season or miniseries, a Rating element should be included. If included, Rating must contain the highest (most restrictive) rating of episodes in the season.

6.4 Director's Commentary and other track combinations

A Presentation contains all tracks conformed to play together. In a sense it is the tracks that *can* play together. However, it is not necessarily the tracks that *should* play together.

In particular, it is important that the subtitles are correctly matched to audio. It doesn't make sense to play commentary audio and non-commentary subtitles.

To handle this, tracks are grouped in Presentation/TrackMetadata. There is one set for each combination of tracks that *should* be played together. TrackSelectionNumber is used to refer to different groups. TrackSelectionNumber='0' is the primary group and should contain at least 'primary' audio tracks. Note that TrackSelectionNumber does not provide metadata on the track selection group—that must be derived from track metadata.

Alternatively, these groups can be assigned their own Presentation and given distinct Audiovisual element or even distinct Experiences.

Note that the user should likely be given the choice of playing track combinations that do not necessarily make sense. Let's say, for example, they want director's commentary subtitles with primary audio. Who are we to say no?

6.5 Organizing Experience

6.5.1 Episodic

6.5.1.1 Episode Ordering

In this best practice, ordering comes from the Experience rather than BasicMetadata. When referencing sequenced episodic assets (episodes, seasons, etc.), ExperienceChild/SequenceInfo must be populated. SequenceInfo/Number must be included.

6.5.1.2 Alternate Episode Ordering

Some shows have different episode numbering in different geographies. To support this, the ExperienceChild element contains sequence information that defines the correct sequencing. Note that the episodes are identical regardless of reordering; including using the same identifiers (i.e., EIDR IDs). As stated in [Manifest], Section 8.3.4, ExperienceChild/SequenceInfo supersedes BasicMetadata/SequenceInfo and ExperienceChild/Relationship supersedes BasicMetadata/Parent/@relationshipType.

Consider the following Season containing a series of episodes. The yellow arrows shows where an ExperienceChild element has referenced an episode. The Episode's BasicMetadata/SequenceInfo defines a default sequence. In this illustration, it orders them as 1, 2, and so forth.

XML might look something like this. There are three episodes called Episode A, Episode B and Episode C. In the US, the sequence is A, then B and then C.

```
<manifest:Experience ExperienceID="md:experienced:org:craig:SeasonX-US" version="1">
    <manifest:Region>
        <country>US</country>
    </manifest:Region>
[…]
<manifest:ExperienceChild>
    <manifest:Relationship>isepisodeof</manifest:Relationship>
    <manifest:SequenceInfo>
        <Number>1</Number>
    </manifest:SequenceInfo>
    <manifest:ExperienceID>md:experienceid:org:craig:EpisodeA</manifest:ExperienceID>
</manifest:ExperienceChild>
<manifest:ExperienceChild>
    <manifest:Relationship>isepisodeof</manifest:Relationship>
    <manifest:SequenceInfo>
        <Number>2</Number>
    </manifest:SequenceInfo>
    <manifest:ExperienceID>md:experienceid:org:craig:EpisodeB</manifest:ExperienceID>
</manifest:ExperienceChild>
<manifest:ExperienceChild>
        <manifest:Relationship>isepisodeof</manifest:Relationship>
    <manifest:SequenceInfo>
```

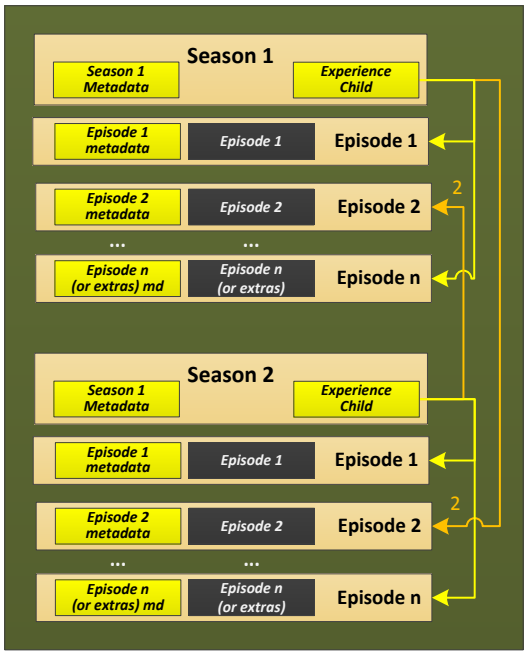
<Number>3</Number>
</manifest:SequenceInfo>
<manifest:ExperienceID>md:experienceid:org:craig:EpisodeC</manifest:ExperienceID>
</manifest:ExperienceChild>

To resequence the episodes, all that needs to be done is add SequenceInfo/Number to each ExperienceChild element in the Season. In the following illustration, episodes 1 and 2 have been swapped.

Adapting the XML from above, everywhere other than the US the first episode is B, the second is A, and the third is C.

<manifest:Experience ExperienceID="md:experienced:org:craig:SeasonX-Not-US" version="1">
<manifest:ExcludedRegion>
<country>US</country>
</manifest:ExcludedRegion>
[…]
<manifest:ExperienceChild>
<manifest:Relationship>isepisodeof</manifest:Relationship>
<manifest:SequenceInfo>
<Number>1</Number>
</manifest:SequenceInfo>
<manifest:ExperienceID>md:experienceid:org:craig:EpisodeB</manifest:ExperienceID>
</manifest:ExperienceChild>
<manifest:ExperienceChild>
<manifest:Relationship>isepisodeof</manifest:Relationship>
<manifest:SequenceInfo>
<Number>2</Number>
</manifest:SequenceInfo>
<manifest:ExperienceID>md:experienceid:org:craig:EpisodeA</manifest:ExperienceID>
</manifest:ExperienceChild>
<manifest:ExperienceChild>
<manifest:Relationship>isepisodeof</manifest:Relationship>
<manifest:SequenceInfo>
<Number>3</Number>
</manifest:SequenceInfo>
<manifest:ExperienceID>md:experienceid:org:craig:EpisodeC</manifest:ExperienceID>
</manifest:ExperienceChild>

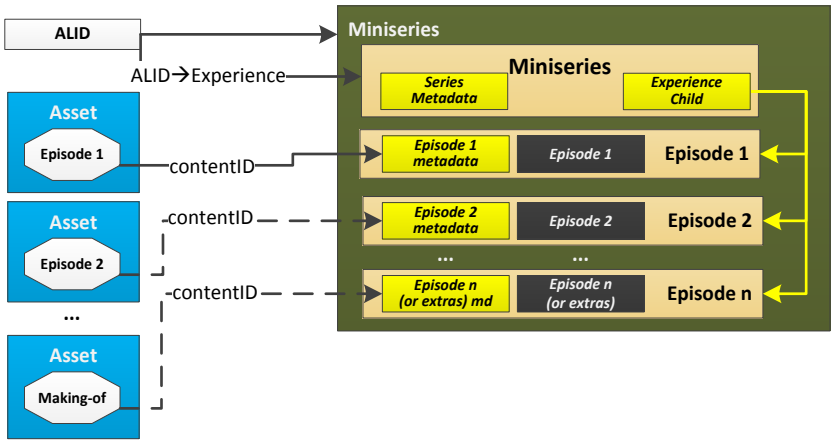
Similarly, Episodes can be moved between seasons. In the following illustration, Episode 2 from Season 1 has been swapped with Episode 2 of Season 2.



6.5.1.3 Miniseries

Miniseries are organized like Series, but child references are to episodes rather than seasons. This is illustrated in the following figure:

Within the Miniseries Experience, BasicMetadata/WorkType is 'Series'. Note that, ExperienceChild/Relationship is still 'isepisodeof'.



6.5.1.4 Series (or other parents) of Alternately Numbered Seasons

A given instance of a season or miniseries can organizes episodes in the correct order. This section describes how to organize those seasons or miniseries into a parent object.

Seasons with a particular episode order will be regionalized. That is, the Experience/Region or Experience/ExcludedRegion elements will be populated to indicate where that ordering is to be used. This means there are multiple season or miniseries instances.

Generally, miniseries are the top-level object so no further action is required. If it is not the top level object then use the series rules below.

Series must reference and ordered sequence of seasons. This is done with ExperienceChild elements with SequenceInfo/Number dictating the order. In the case of multiple episode ordering, there are two options

- Encode a distinct Experience for each regionalized Series. This can undesirable because a new Season Experience is required if any episode is reordered anywhere across the series.
- Include an ExperiencChild within a single Series Experience for each ordered season. If a particular season has multiple orderings, the Series will include an ExperienceChild instance for each. This means that there can be multiple instances of ExperienceChild with the same ExperienceChild/Sequence/Number. Those interpreting the Series Experience must select the appropriate Season Experience for the applicable region given that Season's Experience/Region or ExcludedRegion.

6.5.2 Trailers

6.5.2.1 Default Trailer

In some conditions, there are several applicable trailer options but one is intended as the primary or default trailer. For example, the default trailer is typically used as the primary trailer for sales purposes. The default trailer should be used when there are no matching language or region tags.

To designate a trailer as the default trailer, set Audiovisual/SubType to "Default Trailer" as defined in Section 8.2.2.

6.5.2.2 Country-specific Trailers

When a trailer is designed for specific country, this can be indicated by setting BasicMetadata/CountryOfOrigin to the country code for that country.

ReleaseHistory can also be encoded with all countries where a trailer would apply by setting ReleaseHistory/DistrTerritory. For the purpose of selecting a trailer for a given country, all fields other than ReleaseHistory/DistrTerritory can be ignored.

7 FILE DELIVERY

7.1 File Manifest

The File Manifest is an XML document that describes the structure of files delivered from one party to another. [Manifest] describes how to encode the FileManifest element. This section provides additional guidance on how to use the File Manifest in various applications.

7.2 Package Concept and Identifier

A set of files is called a Package and is identified with a PackageID.

Avails must be identified in a globally unique manner. An ~~Avail identifier~~ALID might be used by the studio or various service providers partnering with the studio.

7.2.1 What an Package Identifier Identifies

A Package ID represents a collection of files associated with a delivery.

There are two use cases here

- PackageID applies to a single delivery of files. Any files subsequent delivery would require a new PackageID. Versioning is handled outside the scope of File Manifest.
- Package ID applies to a set of files, regardless of when and how they are delivered. Subsequent versions of the File Manifest, as distinguished by FileManfiest/PackageDateTime, represent the complete set of files that the sender expects the recipient to obtain.

7.2.2 Constructing an PackageID

A Package ID SHALL be globally unique. The same PackageID SHALL NOT be used for distinct sets of files.

A PackageID SHOULD comply with the PackageID format above. This is not a strict requirement, but it will make global uniqueness much easier and avoid us IDs in the wrong context. Note that UUIDs avoid the first issue, but not the second.

An PackageID SHOULD be based on an EIDR ID.

7.2.3 EIDR IDs and PackageIDs

A PackageID using an EIDR IDwould be of one the two following forms:

"md:~~avail~~alid:eidr-s:"<Short EIDR>

"md:~~avail~~alid:eidr-x:"<Short EIDR>":"<extension>

In one usage, an EIDR ID is created for the Package as a Compilation. In this case, the EIDR-s form would be used. Alternatively, the EIDR ID could be constructed as an EIDR-x~~,~~

~~possibly matching the AvailID or possibly extending the AvailID.~~ See Section 3.2 for instructions how to use an EIDR ID for ~~Avail IDs~~ALID.

If an ~~Avail ID~~ALID is further extended, <extension> part would include both the Avail unique information and the package information. For example, let's assume a single title: The Devil is in the Details, EIDR Edit = 1012-7947-21D5-9D24-CC5F-H. ~~AvailIDs~~ALID might be:

```
md:~~avail~~alid:eidr-x:1012-7947-21D5-9D24-CC5F-
H:craigsmovies.com_july_NorthAmerica
```

The Package ID could be:

```
md:package:eidr-x:1012-7947-21D5-9D24-CC5F-H:craigsmovies.com_july_NorthAmerica_pkg1
```

Note that <type> changed to 'package' and '_pkg1' was added to indicate that this Package ID is for the first package (package 1) associated with that Avail.

Identifiers are to be processed in their entirety as an opaque object. Naming conventions in extensions are intended to support uniqueness human readability and SHOULD NOT be parsed automatically to extract information.

7.3 File Identification and Versioning

7.3.1 Identifying Files

Files can be identified in several ways. The most important distinction when identification refers to a specific version (e.g., a particular encoding) or to the contents (e.g., an encoding of a particular language track). For versioning files, it is important to know both.

The most definitive identification of a given file at the bit level is the file Hash (FileInfo/Hash). It is effectively guaranteed to be unique for any file.

The most flexible identification is an identifier (i.e., FileInfo/Identifier). An identifier can identify version and contents. Multiple identifiers can be included.

7.3.2 Versioning

Files versioning requires knowing two files are the same except for the version. They must have some level of identification, but also be different in some manner.

Currently the following methods are available for versioning.

- FileInfo/Version element
- Information inherent in the file
- Determine from Media Manifest which files are relevant

7.3.2.1 FileInfo/Version FileInfo/FileDate versioning

The simplest means to determine the most recent version is the use of the Version element in FileVersion. The initial has Version either absent or '0'. These are semantically equivalent. A Version>0 means there has been an update. The file with the largest Version is the most current.

As this is the simplest and most unambiguous, this method is recommended.

FileInfo/FileDate can also be used to determine file version.

7.3.2.2 Information in the File

Some files contain version information in the file. This is not the easiest method to determine version and it does not apply to all files, so it is not the preferred method.

Version information in files include:

File Type	Means to determine version
Metadata	Basic Metadata such as found in Media Entertainment Core (MEC) can be versioned using CoreMetadata/Basic/UpdateNum
Media Manifest	Media Manifest is versioned using MediaManifest/@updateNum. ExtraVersionReference can be used as an additional versioning identifier.
Avails	Avails can be versioned using Avail/Disposition/IssueDate. Note that IssueDate can include time.
Common File Format (CFF)	Common File Format files can be versioned by APID with additional information in MetadataMovie/ContainerVersionReference and MetadataMovie/@MetadataVersionReference.

7.3.2.3 Media Manifest

The Media Manifest can link Avails to Experience and ultimately to media files. In general, if a file appears in the Media Manifest it is required.

If the Manifest is complete, there is no ambiguity about what file is what or how it is used in the system. Media file information is in the Inventory. Metadata is referenced by ContentID.

7.4 File Delivery

The File Manifest design recognizes that not all files are necessarily delivered at once and may be updated once delivered. Methods are provided for generating and accepting incremental deliveries.

7.4.1 Delivery Methods

The File Manifest supports most common means of delivering files: FTP, email, HTTP GET, etc. These methods are defined in [Manifest]. This document has no further recommendations regarding delivery methods.

7.4.2 Delivering Sets of Files

The File Manifest supports delivering multiple files together. To do so, include multiple instances of FileInfo, one for each file.

The basic types of files delivered are as follows, although any type of file can be delivered.

- Manifest file – A file with data described in this section
- Metadata files
- Media files – audio, video, subtitle, apps, images, text, interactivity/apps)
- Avails files
- Ancillary files (any other files). Images are Ancillary files.

All files could be delivered in a single manifest or they could be spread across multiple manifests. It is reasonable to have one of files in one manifest and another set in another manifest. One should consider timing, including updates, when deciding how files should be grouped. Using the New Package Model described below (a new FileManifest for each drop), this is less critical. However, if the Single Package Model is used, the initial choice of grouping must be maintained for updates.

7.4.3 Incremental Delivery

There are two ways to handle incremental delivery:

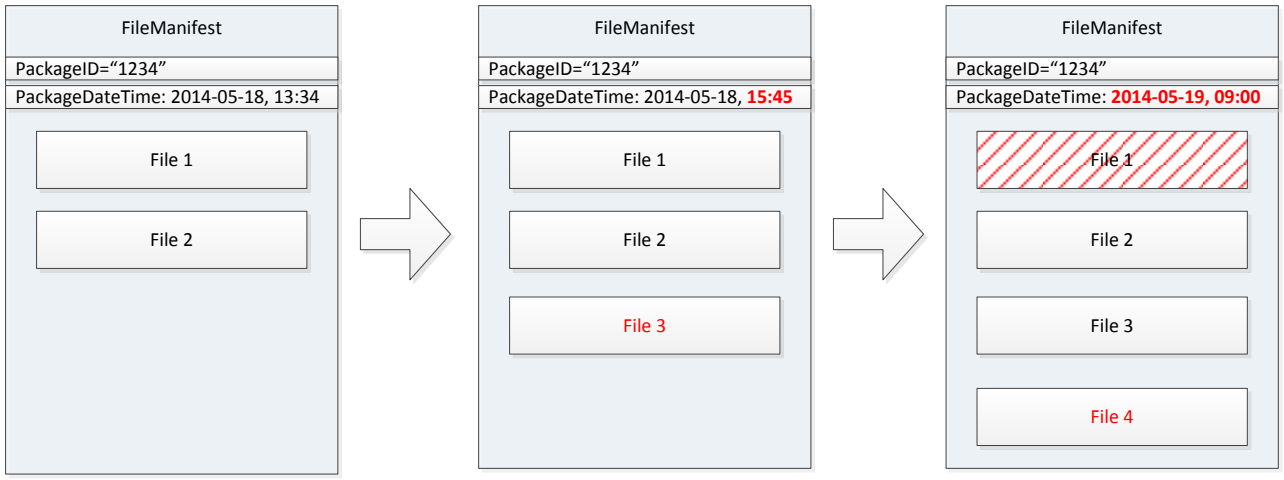
- A single Package definition is used, but updated to reflect changes.
- A new Package is generated for each update.

These are described in the following sections.

7.4.3.1 Single Package Model

The Single Package Model uses the same Package and PackageID for all deliveries. Each Package represents a complete snapshot of the delivery. The goal is for the recipient to have the same files represented in the File Manifest. If a file is added to the File Manifest, that file should be obtained. If a file is removed from the File Manifest, that file can be removed.

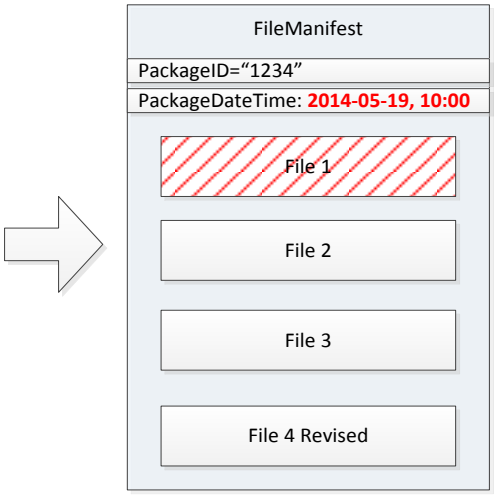
Consider the following. There are three deliveries of the File Manifest. They all have the same PackageID. The date-time increases with each delivery. The second drop adds File 3. The third drop adds File 4 and removes File 1. As the 2014-05-19 09:00 delivery is the last, the recipient should have Files 2, 3 and 4.



The removal of File 1 is indicated by FileInfo/Delivery/DeliveryMethod='removed'.

In the second delivery, File 1 and File 2 have already been delivered. The same is true for File 3 in the third delivery. This can be determined by the recipient by looking at the identification and the Hash. If the file is still deliverable, it should be noted in DeliveryMethod. However, if the file is presumed delivered and is no longer available, it should be noted by setting DeliveryMethod='delivered'.

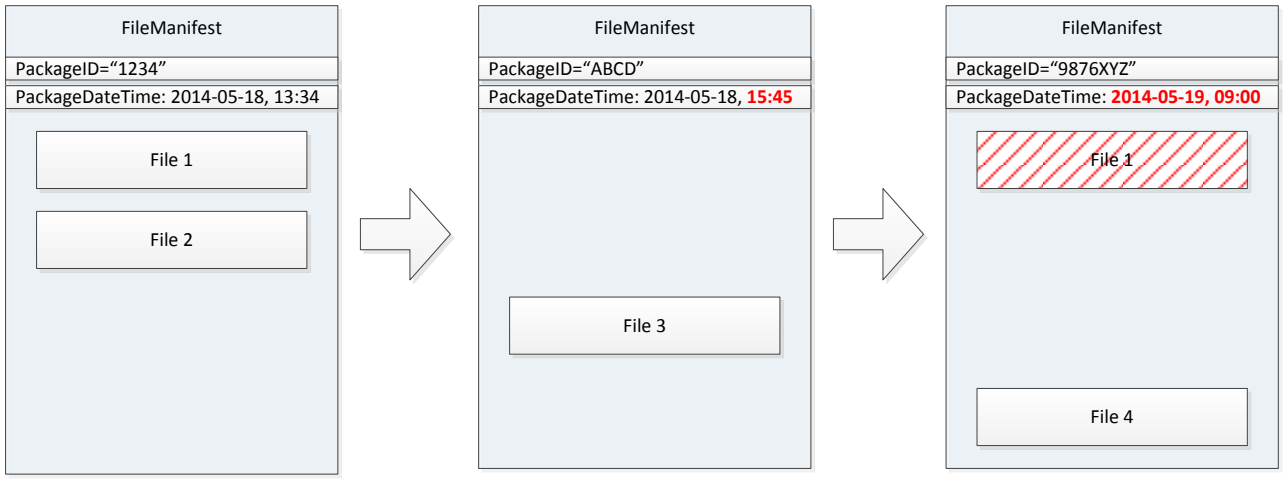
The following diagram that follows the example above illustrates a File update:.



In this example, File 4 is revised. To recognize the File 4 is updated is must have the same Identifier as the previous version. Version is updated to be a higher value than the previous Version.

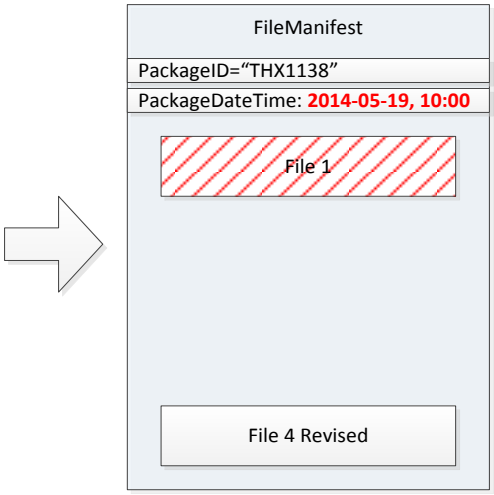
7.4.3.2 New Package Model

In the New Package Model, each File Manifest has a unique PackageID. Only new or removed files are included. The following example has the same net effect as the example in the Single Package Model.



This model is requires every update to processed, and processed in order.

Update is similar in the model.



Like the previous model, Identifier remains constant and the Version is revised.

7.5 Verifying File Correctness

It is strongly suggested that Hash be included and verified. Hash is required in cases where Hash is used to differentiate versions.

Software is subject to manipulation and should be treated as sensitive. All files that include any code, whether source or executable, must include Hash in the File Manifest and must be checked.

XML canonicalization is not required. Two XML documents might be functionally equivalent, but have slightly different encoding (e.g., different white space). There is no attempt to match such equivalences so XML Canonical Form [XML-C] is not required.

Hash algorithms (Hash/@method as defined in [CM], should be either 'MD5', 'SHA-1' or 'SHA-256'. It is strongly recommended that implementations ingesting data from a File Manifest be capable of verifying using these algorithms.

If Hash is used, it is strongly recommended that the Length element be used to ensure that the sender and the receiver are checking the correct number of bytes. If Length differs more than a few bytes from the actual length of the file, be sure to verify that the hash was properly generated (think Heartbleed).

8 OTHER ELEMENT ENCODING RULES

8.1 Use of Region

Both Avails and the Experience allow region to be encoded. More specifically, the applicable elements are Avails/Transaction/Territory and TerritoryExcluded; and MediaManifest/Experience/Region and ExcludedRegion. These are encoded in accordance with [CM], Section 3.2.

8.1.1 Avail Territories

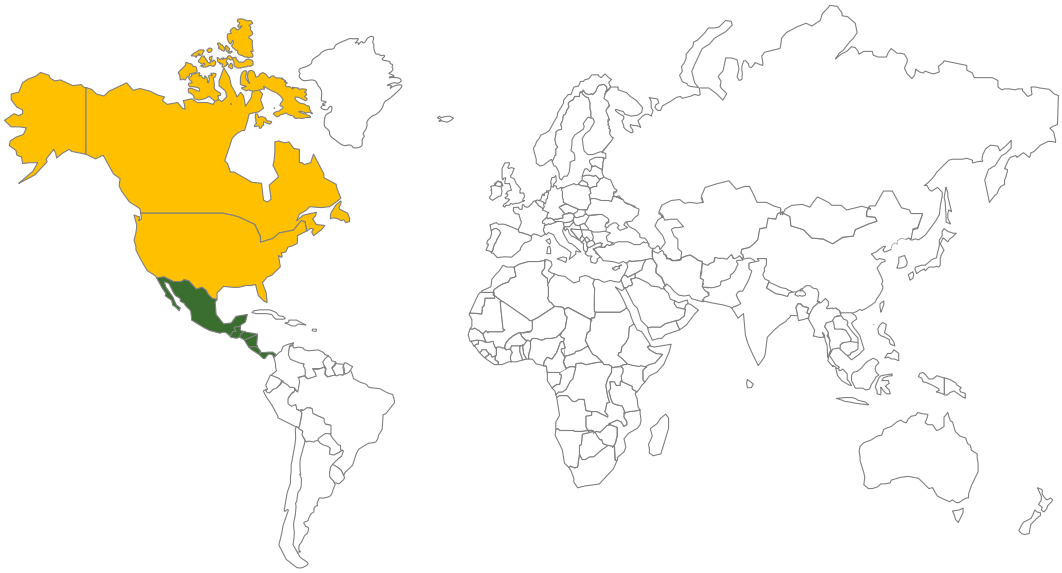
A Transaction element can have multiple Territory elements, so any combination of territories can be covered by an Avail. Furthermore, multiple Transaction elements can be included, expanding the scope of the Avail. Be careful to ensure that no two Transaction elements cover the same territory unless the terms are unique (e.g., SD vs. HD).

Although Avails allows the inclusion of both Territory and TerritoryExcluded, this should not be used. The one exception is when 'Domestic' or 'International' is used, although these should not be used in Avails. Be specific and list countries. This will avoid confusion and potential contradictory information.

Consider the following: Avail 1 is US and Canada, shown in Orange, and Avail 2 is Mexico and parts of Central America shown in Green.

Avail 1: Territory='US', Territory='CA'

Avail 2: Territory='MX', Territory='GT', Territory='BZ', Territory='HN', Territory='SV', Territory='HN', Territory='NI', Territory='HN', Territory='CR', Territory='PA'



The following examples illustrate the use of ExcludedTerritory.

Avail 3: ExcludedTerritory='US', ExcludedTerritory ='CA', ExcludedTerritory ='MX', ExcludedTerritory ='GT', ExcludedTerritory ='BZ', ExcludedTerritory ='HN', ExcludedTerritory ='SV', ExcludedTerritory ='HN', ExcludedTerritory ='NI', ExcludedTerritory ='HN', ExcludedTerritory ='CR', ExcludedTerritory ='PA'

Avail 1, 2 and 3 collectively cover the entire world, excepting Cuba and North Korea.

8.1.2 Experience Regions

The use of Region and ExcludedRegion is analogous the Avail's Territory and ExcludedTerritory.

Note that Experiences are defined to address both region and language. So, there can be multiple experiences for a given region, each with their own language. The following scenarios assume the same language.

First Experience regions should not overlap. If they do, it should be assumed Experiences will be used the following precedence:

- If Region includes the territory in question, this Experience should be used.
- Otherwise, if ExcludedRegion does not include the territory in question, this Experience should be used
- Otherwise, if an Experience does not include Region or Excluded Region, this Experience should be used
- In the above cases, if there is more than one match, use the first Experience listed. If no Experience matches territory, the Experience to use is undefined; and, one should investigate.

Consider the following cases:

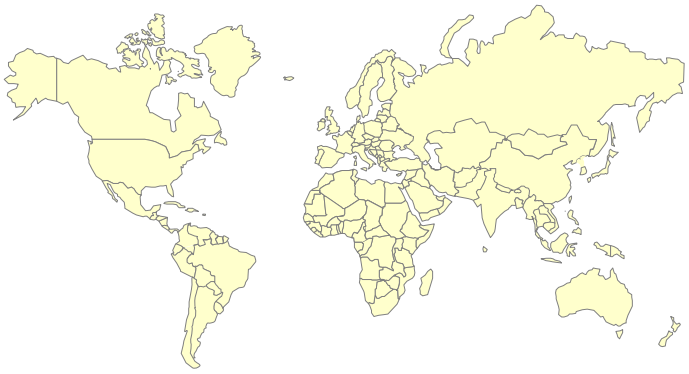
- The Experience has no Region or RegionExcluded, meaning the Experience covers the entire world.

  This is always acceptable. It is presumed that a consumer will not be playing content outside of the Availed territory. So, it does not matter if the scope of the Experience is broader than the scope of the Avail.

- The Experience covers a Region that is a subset of the Avail.

  This is always acceptable. However, it is important that there exists exactly one Experience that covers each part of the territory.

Using the Avail 1 example above (US and Canada), each of the following are acceptable (although not simultaneously):



One Experience worldwide (no Region or RegionExcluded)

One Experience for US and Canada together (Region='US', Region= 'CA')

Two Experiences: One for US (Region='US') and one for Canada (Region= 'CA').

8.2 Audiovisual Type/SubType

The Audiovisual element contains Type and SubType elements to provide information about the referenced Playable Sequence or Presentation.

The Media Manifest specification defines Type as one of the following:

- 'Main' – Main title (typically the feature)
- 'Promotion' – Trailers, teasers, etc.
- 'Bonus' – Additional material related toward the Main Program, such as, deleted scenes, making-of, etc.
- 'Other' – Any other material included

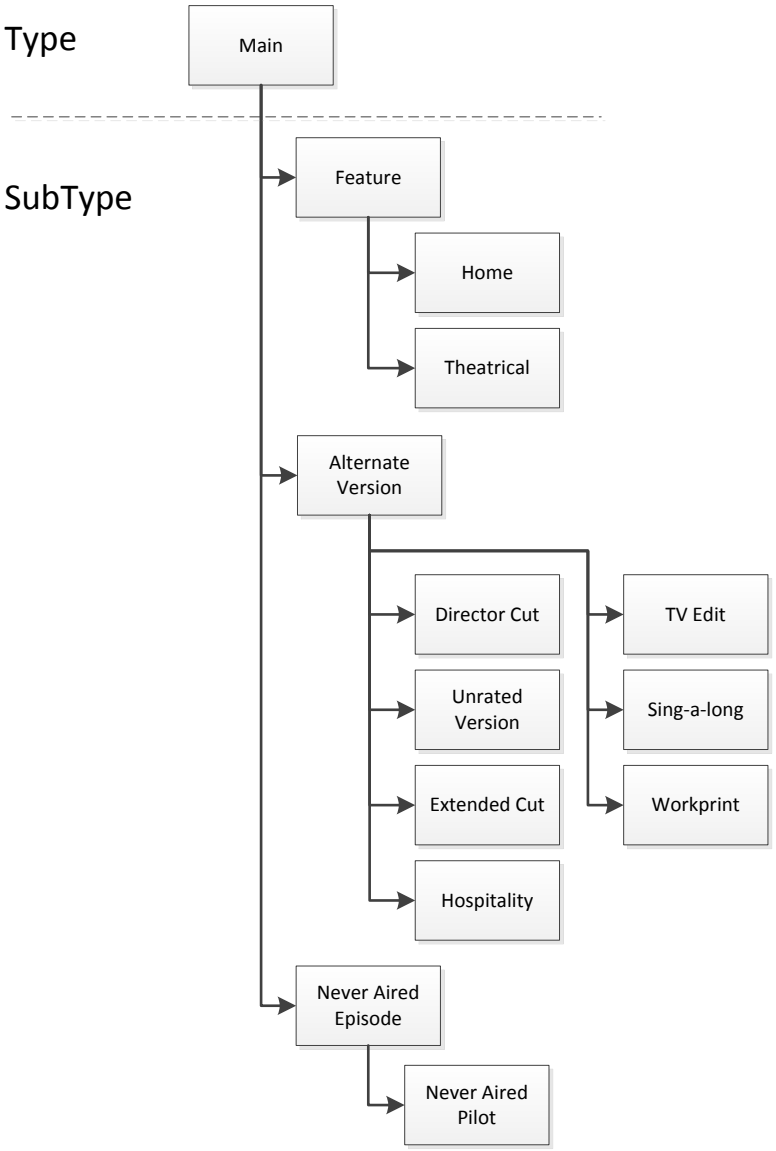
This section provides guidance on encoding SubType. Zero or more SubType elements can be included[1]. Generally, SubType values are relevant only to specific Type values. For example, the "Trailer" SubType only applies to the "Promotion" Type. When more than one SubType is included, subsequent SubType instances modify the first SubType. This is not a hard and fast rule, and any applicable SubType should be included.

The following sections define the SubType values applicable to each Type value. In each section, a diagram is provided showing relationships and a table is provided describing each value.

Encoding must be as given (including spaces and dashes). It is recommended that interpretation is case insensitive.

[1] Version 1.0 allowed only one SubType instance, but this is corrected in later versions.

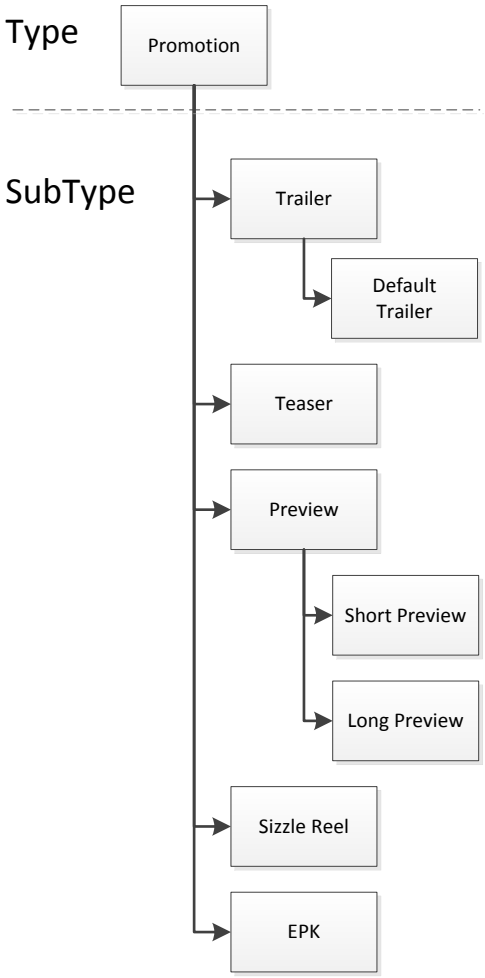
8.2.1 Main Type



SubType value	Description	Parent
Feature	Main feature. This avoids any ambiguity that this is the primary asset (movie, TV episode, etc.) in the collection.	
Home	Home entertainment version. This is assumed unless otherwise stated.	Feature
Theatrical	Theatrical version. This may also appear under Alternate Version if desired for presentation or marketing reasons.	Feature
Alternate Version	Indicates another version of the asset	

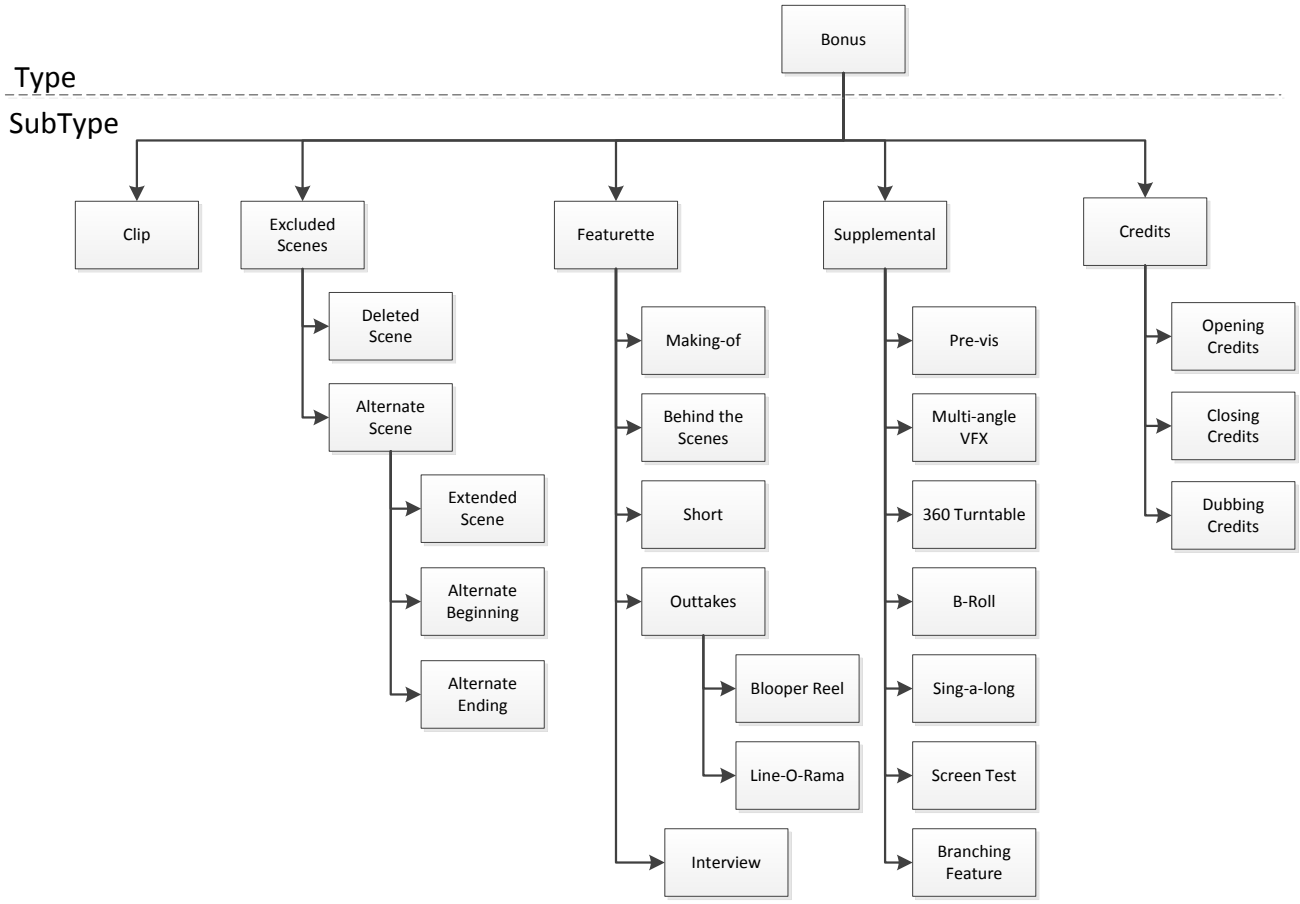
Director Cut	Director's cut	Alternate Version
Unrated Version	Unrated version, typically with an implied rating higher than the original (e.g., "Unrated Edition").	Alternate Version
Extended Cut	Extended version of the work, not including director's cuts.	Alternate Version
Hospitality	Version created for hospitality (airplane, hotel, etc.)	Alternate Version
TV Edit	Version edited for television	Alternate Version
Sing-a-long	Full feature sing-a-long. Note that sing-a-long can also be Bonus.	Alternate Version
Workprint	Post-production version of the complete film.	Alternate Version
Never Aired Episode	TV Episode that was not aired. This would typically be included in a list with episodes that were aired.	
Never Aired Pilot	The episode than never aired is a pilot.	Never Aired Episode

8.2.2 Promotion Type



SubType value	Description	Parent
Trailer	Short-form promotional content	
Default Trailer	Use this one as the primary trailer when showing title. (e.g., presale)	Trailer
Teaser	Short trailer that doesn't show a lot of footage, builds buzz	
Preview	Preview	
Short Preview	Short preview. This generally refers to the first 2 minutes of a work	Preview
Long Preview	Long preview. This generally refers to the first 10 minutes of a work	Preview
Sizzle Reel	"A sizzle reel is a fast paced video that is short and incorporates creativity with sounds and engaging sights to advertise a product or any concept on TV" (http://prsizzlereel.com/sizzlereel/what-is-a-sizzle-reel/)	
EPK	Electronic Press Kit. Container for clips/other promotional material given to the press	

8.2.3 Bonus Type



SubType value	Description	Parent
Clip	Subset of a Type="Main" asset. If is not part of a main asset, is would generally be an Excluded Scene	
Excluded Scenes	Any material that did not appear in the original Type="Main" asset.	
Featurette	General category that can include shorts, making-of's, and other types of film-related content	
Supplemental	Additional finished material that would not be considered a Featurette.	
Credits	Credits	

Excluded Scenes

SubType value	Description	Parent
Deleted Scene	Scene deleted from final cut	Excluded Scenes
Alternate Scene	A scene that replaces a scene in the final cut, but is not considered an Extended Scene	Excluded Scenes
Extended Scene	Alternate scene that extends ~~ana~~ scene in the final cut.	Alternate Scene
Alternate Beginning	Scene that represents a different beginning.	Alternate Scene
Alternate Ending	Scene that represents a different ending.	Alternate Scene

Featurette

SubType value	Description	Parent
Making-of	Documentaries that address some aspect of creating a work. These may address entire works, specific scenes, special effects, costumes or any other aspect of filmmaking.	Featurette
Behind the Scenes	Behind the scenes.	Featurette
Short	Short that accompanies a main title. For example, an animated short associated with an animated film.	Featurette
Outtake	Takes that are not included in the final version.	Featurette
Blooper Reel	Composition of outtakes from filming, usually edited together quickly to increase comedic value	Outtake
Line-O-Rama	Composition of different lines, typically ad-lib lines, from a work. Similar to a gag reel	Outtake
Interview	Interviews with cast/crew about a role or movie, generally released as promotional content	Featurette

Supplemental

SubType value	Description	Parent
Pre-vis	Footage of a scene before final visual effects have been processed. Usually batched with clips from several VFX stages to show a progression	Supplemental
Multi-angle VFX	Multi-angle representations of visual effects	Supplemental
360 Turntable	View of object from all angles (360 degrees)	Supplemental
B-Roll	B-roll material collected to fill in scenes. This might be included in the final cut, or not.	Supplemental
Sing-a-long	Scenes sing-a-long visuals. If it is a complete work, it should be an Alternate Version	Supplemental

Screen Test	Footage from actor's auditions for a role	Supplemental
Branching Feature	Commentary is included in divergent paths that the user can navigate.	Commentary

Credits

SubType value	Description	Parent
Opening Credits	Credits shown at the beginning	Credits
Closing Credits	Credits shown at the end	Credits
Dubbing Credits	Credits for audio dubbing in particular languages. Can be used in conjunction with Opening Credits or Closing Credits.	Credits

8.2.4 Other Type

No SubType values are currently defined for "Other".

8.2.5 Studio-specific Types

For other types not covered, the following syntax is recommended:

"Other:"+[<org>"+":"]+<label>

Where <org> is an organization-specific name and <label> is a unique label. Note that <org>+":" is optional.

For example, if Warner Bros wished to add a Maximum Movie Mode Subtype, it would look like this: `Other:Warner:MMM`.

8.3 Language Tags

Language tags can be confusing, but once you know the basic rules they are pretty straightforward.

A language tag is constructed using the following (from RFC 5646):

```
langtag       = language
                ["-" script]
                ["-" region]
                *("-" variant)
                *("-" extension)
                ["-" privateuse]
```

The details of each part are described in RFC 5646. As noted above, there must always be a `language` part; for example, 'fr' for French and 'en' for English. The `region` field is also commonly used, for example, 'fr-CA' (French, Canada) to represent Québécois. These other parts

are called subtags. Language and subtag values can be found in the IANA Language Subtag Registry at http://www.iana.org/assignments/language-subtag-registry/language-subtag-registry.

For example, within the registry, you could find:

```
Type: language
Subtag: fr
Description: French
Added: 2005-10-16
Suppress-Script: Latn
```

and

```
Type: region
Subtag: CA
Description: Canada
Added: 2005-10-16
```

These are the entries that respectively correspond with 'fr' and 'CA' in 'fr-CA'. Note that `Type` corresponds with the definition of langtag from RFC 5646.

[Manifest] Section 1.3.2 provides specific instruction on how to match language tags. The same logic applies to matching other instances of language tags such as matching a user's preferred language to an Experience.

Generally speaking, it is best not to be overly specific with language tags encoded in Avails or a Manifest unless complementary languages are provided. For example, if only one French language track is provided it is assumed that it will be used in France, Canada, Switzerland and other French speaking countries; so, it is best to encode it as 'fr' rather than 'fr-FR'. However, if multiple languages were provided, then be specific to differentiate them.

As a rule, where there are multiple language tags using the same language subtag, use the best match ('fr' matches 'fr' better than 'fr-CA' and 'fr-CA' matches 'fr-CA' better than 'fr').

ANNEX A ORDERING FILES (DRAFT)

This section provides recommendations on ordering. If it is sufficiently general, it could become its own specification. It is envisioned that API and data definitions will follow.

A Retailer must generally inform the Distribution Entity which materials it wants.

Following could be supplied by a Retailer for ordering:

- Preferences
  - Type of files preferred in order of preference
  - Priority for encodes (e.g., 5.1 is preferable to 2.0)
- Minimal expectations
  - Which set of components is minimally required for a region

The order itself could be comprised of the following

- Identification of content required, suggestion identification are as follows: The Distribution Entity should be able to handle both ALID and File identification.
  - ALID – Logical Asset identifies the set of content. The Distribution Entity should know how to map an ALID to content files.
  - Files – Individual files can be requested. This is particularly true if a file was missing or corrupted.
  - ~~AvailID – Identifies the associated Avail. This assumes The Distribution Entity is cognizant of Avails. This is rare, so this is not preferred.~~
  - Note: We formerly use 'AvailID', but have since simplified the system by identifying an Avail without this additional ID. Avails are uniquely identified by the combination of ALID and Licensor..

Messages

- Readiness should be reported, including approvals and rejections
- An automated ordering system should have the capability to indicate that human inspection of the request is required.