Cross Platform Extras: CPE-Manifest

CONTENTS

REVISION HISTORY

Version	Date	Description
1.0	April 15, 2016	Initial publication

1 INTRODUCTION

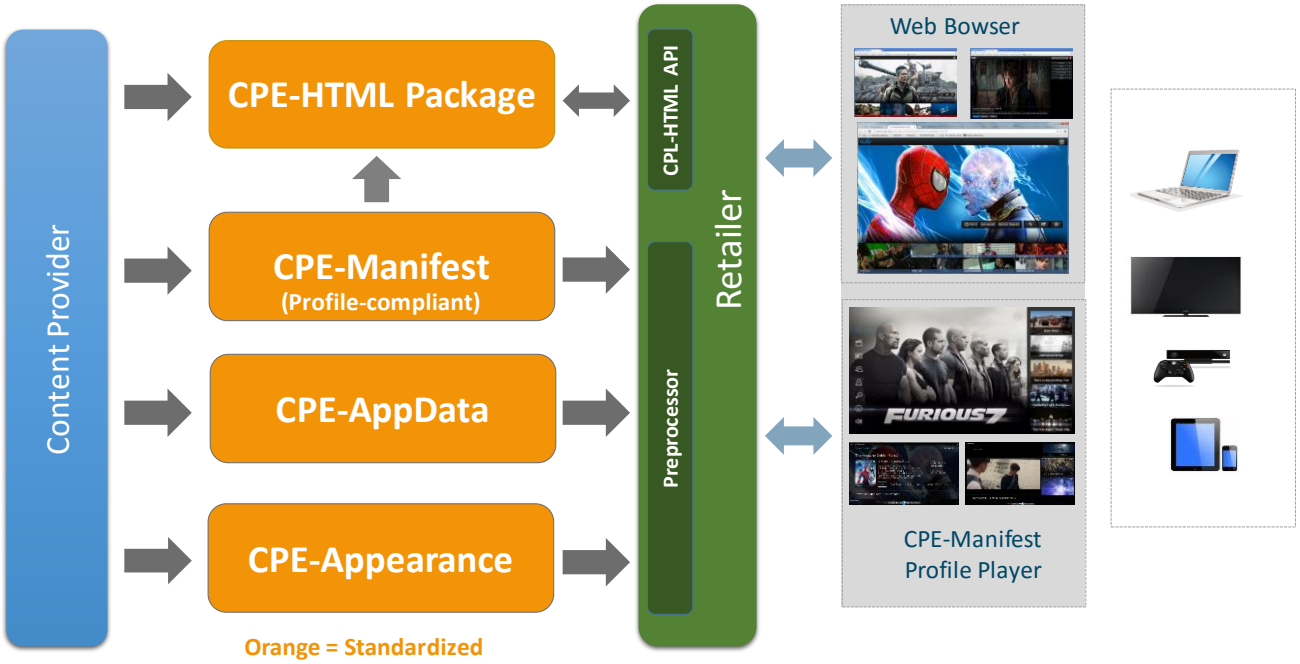
This document describes Cross Platform Extras Manifest (CPE-Manifest), defining how to use Common Media Manifest and to create a portable interactive experience.

This document assumes familiarity with the referenced specifications, particular *Common Media Manifest Metadata*. In many cases, this document builds on the best practice for delivery called *Using Media Manifest, File Manifest and Avails for File Delivery (Best Practices)*.

1.1 Background

The MovieLabs Common Manifest allows complex user experiences to be assembled from individual assets, e.g. a video track, audio track, subtitle track, images, applications and more. It forms the basis for other aspects of the workflow including product definition and component based asset delivery.

This document provides specific instructions on using and interpreting Media Manifest documents to describe content structure, bonus material, metadata, applications and other aspects of a consumer experience. The experience defined by Media Manifest fully describes content and its relationship, but leaves the presentation (user interface) to the player designer.



1.2 Document Organization

This document is organized as follows:

1. Introduction—Provides background, scope and conventions
2. Using this Document – Recommendations for document use

3. General Guidance – General rules for using Media Manifest for interactivity
4. Manifest Construction Models – Base models on which specific Profiles are based
5. Manifest Encoding Constraints – Specific encoding instructions for various elements and attributes
6. Annex: Profiles – Specific Profiles for interactivity Manifests

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 v1.5, TR-META-MMM, www.movielabs.com/md/manifest
[Avail]	EMA Content Availability Data (Avails), TR-META-AVAIL, www.movielabs.com/md/avails
[Delivery]	BP-META-MMMD, *Using Media Manifest, File Manifest and Avails for File Delivery (Best Practices).* www.movielabs.com/md/manifest
[IMFDelivery]	TR-META-MMMIFM, *Using Common Media Format with Interoperable Media Format*, www.movielabs.com/md/manifest
[MEC]	Media Entertainment Core, TR-META-MEC, www.movielabs.com/md/mec
[MMC]	Media Manifest Core, TR-META-MMC, www.movielabs.com/md/mmc
[CPE-HTML]	Cross-Platform Extras API, TR-CPE-API, www.movielabs.com/md/cpe
[EIDR-UG]	EIDR 2.0 Registry User's Guide, eidr.org/technology/
[EIDR-ID]	EIDR ID Format, eidr.org/technology/

1.5 Informative References

[CMP]	Common Media Package, www.uvcentral.com

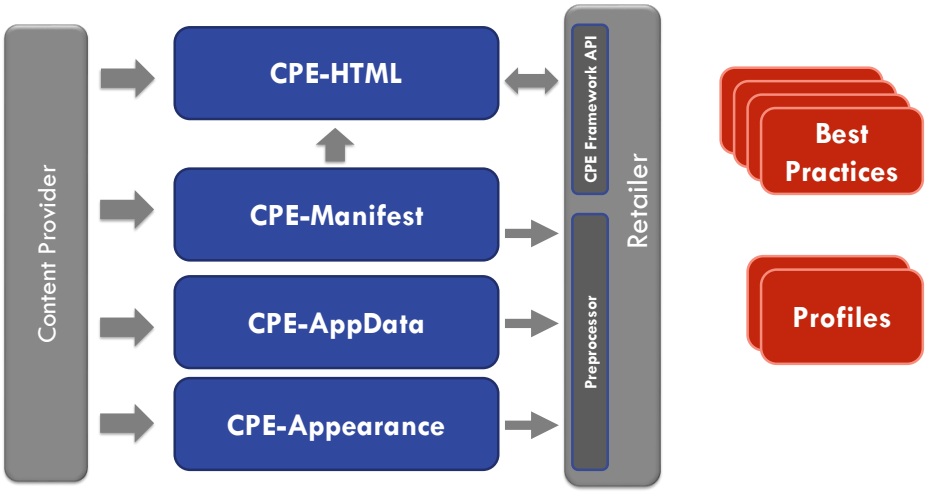
[DOI]	Digital Object Identifier (DOI), www.doi.org
[XML-C]	Canonical XML, Version 1.0. http://www.w3.org/TR/xml-c14n
[ISO26324]	ISO 26324:2012, *Information and documentation -- Digital object identifier system*
[EIDR-V]	How to Use EIDR to Identify Versions for Distribution Purposes: Edits, Languages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

2 USING THIS DOCUMENT

This document is targeted to people who are implementing interactivity using CPE-Manifest. The primary focus is on controlled uses of Media Manifest for supporting a constrained user experience. For the simple delivery of Trailer or Bonus Material with a title, see Media Manifest Core [MMC] and section 2.3.2 of Best Practices for Delivery [Delivery].

The Media Manifest is very general and supports many applications. However, it would be impractical to take an arbitrary Media Manifest and build a user experience around it. Our goal is to provide an outstanding user experience while keeping both authoring and playback as simple as possible. So, this document describes rules (constraints) for authoring Media Manifest. From this, players can be written.

Documentation is based on the following model. Each standardized object corresponds with a spec [CPE-HTML], this document (elsewhere referred to as [CPE-Manifest]), [CPE-AppData] (pending), and [CPE-Appearance] (pending).



There are two levels of definition

- Models – this defines general rules for a set of Profiles.
- Profiles – very specific rules for one specific use. Ultimately, every interoperable implementation must comply with a Profile.

This document defines one Model and two Profiles.

If the reader is looking to implement a specific Profile we suggest reading the rest of this section for background, then jumping to the specific Profile in the Annex. The Profiles refer back to earlier sections that provide additional rules. The context of the Profile will aid in the understanding of the model. Note that implementation of a Profile will ultimately require reading of the entire document.

This document is intended to be used with Media Manifest Core [MMC] and Best Practices for Delivery [Delivery]. A properly constructed Manifest built to the delivery best practices defines the relationships between the various media components (main title, trailers, bonus material, etc.). This document supplements that definition to help a Manifest fit into a given interactivity profile.

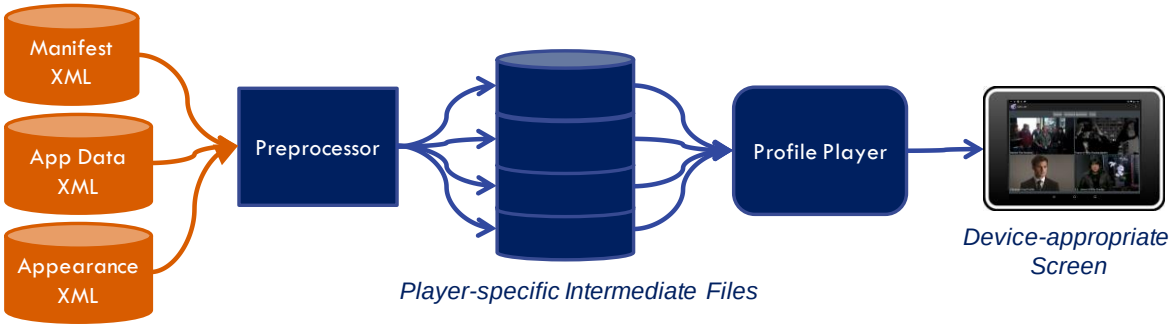
2.1 Using Media Manifest to Support Interactivity

To use the Media Manifest in an interactive application it is necessary to author Media Manifests in accordance with rules (defined here), and playback that Manifest on a "Player". What that Player does with the Manifest is defined by the Profile.

2.1.1 Media Manifest Player

Media Manifest is played through what we refer to here as a "Profile Player" (or "Player"). This Profile Player can play data associated with one or more CPE-Manifest Profiles. It is not anticipated that any Profile Players will execute the XML directly.

Implementations will almost certainly prefer a format other than XML for execution within their player (CPE or otherwise). It is assumed that these implementations will preprocess the XML into a format more appropriate to their environment. For example, we expect many applications, especially JavaScript-based apps, will prefer JSON. Preprocessing can also apply to media assets as well. MovieLabs sample implementation, found at test.movielabs.com/cpe provides an example of preprocessing Media Manifest XML into JSON then using that JSON in a player. The following illustration shows how XML would be translated into a form suitable for the implementation.



The Media Manifest, and optionally App Data and Appearance Data, tells the player about the content (metadata), how it is organized, how it is to be played and where to find it. The Preprocessor takes those instructions and translates it into a form that the Profile Player can interpret create the user experience.

Media assets are referenced in the Media Manifest Inventory (media tracks, images, apps, etc.). This spec is not specific on whether these are access directly from the Profile Player or are also preprocessed (ingested) into the retailer's environment.

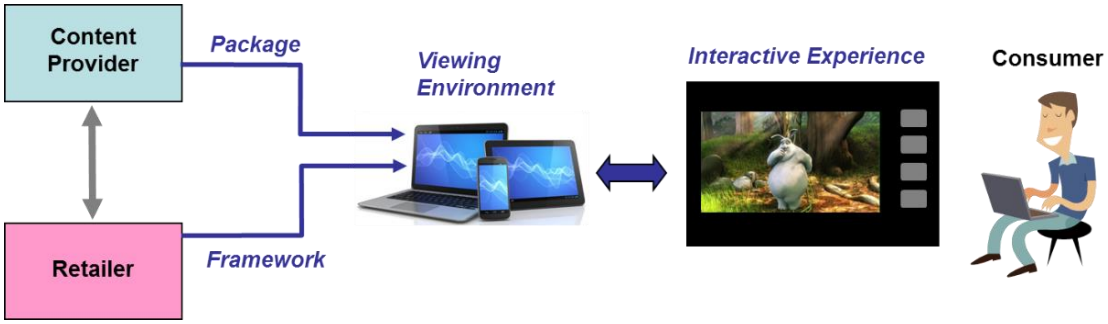
Experiences, media assets and other assets can be downloaded. Currently, there is one complete solution for his model, based on the Common Media Package (CMP), documented in [CMP]. Note that when playing from a downloaded file, playback can be from downloaded files,

online files or both. This gives the flexibility of using streaming for components that are not desired in the CMP (e.g., to save space).

2.1.2 CPE-HTML Package as a CPE-Manifest Player

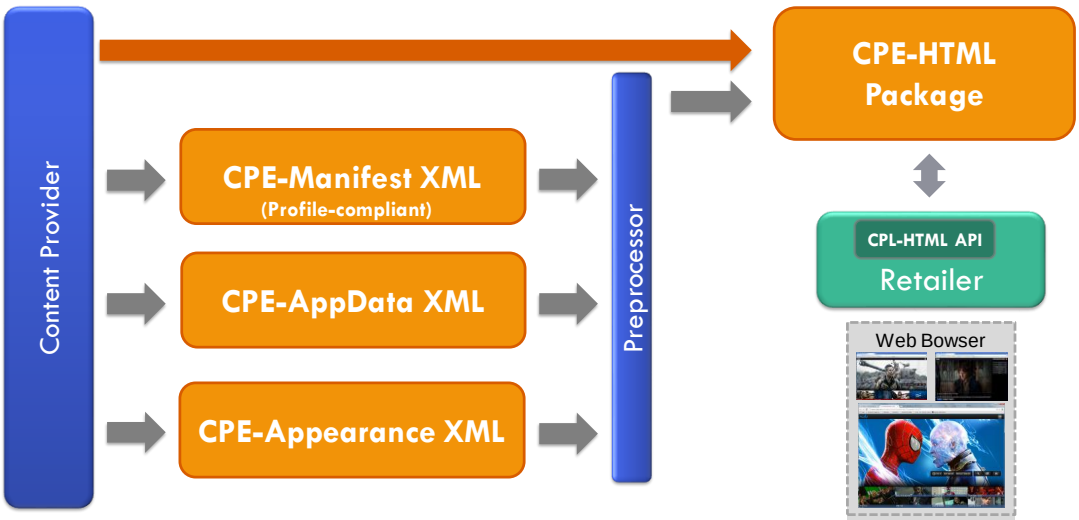
One option for playing content using a Media Manifest is HTML. CPE-HTML defines how to integrate authored content into a retailer environment. In this model, the Media Manifest is input to an CPE-HTML Package (possibly with pre-processing).

The CPE model is as illustrated here:



The CPE-HTML API facilitates the coordinated efforts of both content producers and content retailers/distributors in the creation of interactive viewing experiences. [CPE-HTML] defines a language-neutral interface between the retailer-supplied components (referred to as the "Framework") and the content producer's components (referred to as the "Package"). A full description is [CPE-HTML].

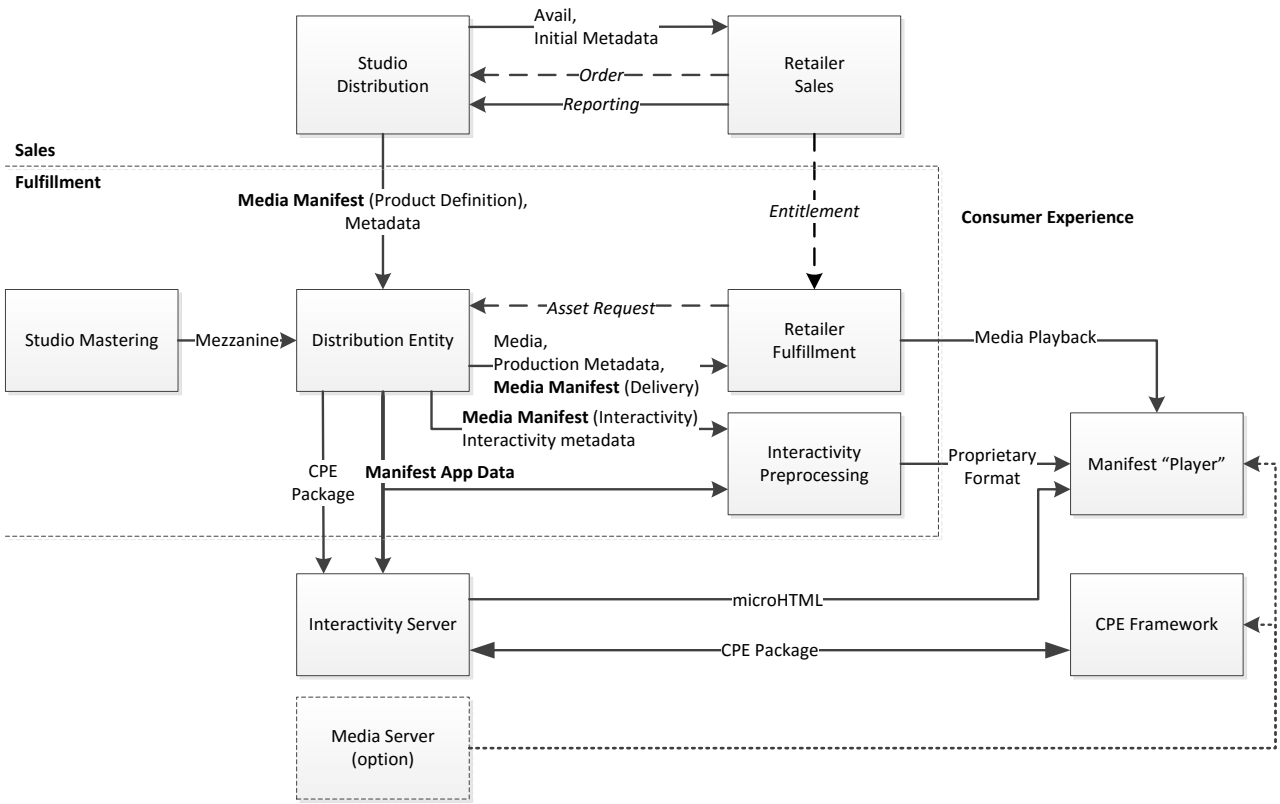
The CPE-HTML Package would be implemented as a generic HTLM/JavaScript Manifest Profile Player. This player would read the data generated by a preprocessor, much as it would for a native Profile Player. Note that in this model, the Content Provider preprocesses the XML files into a format suitable for its authored CPE-HTML Package.



Note that CPE-HTML can be independent of the Manifest and the Manifest can be independent of the CPE-HTML. But, in the model we are discussing here, the CPE-HTML is driven by the Media Manifest.

2.2 Related Material

The Player needs a Media Manifest, media (audio, video, images, apps), and potentially entitlement information. Although there are many ways to deliver these pieces, there are supporting specifications that are fully compatible with this specification. We believe that using these specifications will ultimately simplify the end-to-end process, illustrated in the following:



This model has the following participants. Note that these are roles and do not necessarily align exactly with corporate entities. Although, this document focuses primarily on the Media Manifest input to the Retailer, the referenced documents cover the other areas.

- 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 orders based on an Avail.
- 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. See [Manifest].
- IMF – Interoperable Media Format useful as a mezzanine file [IMFDelivery]
- 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].

3 GENERAL GUIDANCE

3.1 Packaging Content into Media Manifest 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.

3.2 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. A detailed discussion of identifiers is provided in [Delivery], particularly Section 3.

4 MANIFEST INFORMATION MODEL

This section provides additional guidance on Media Manifest construction. All recommendations in [Delivery], Section 6 apply except as noted below.

The Media Manifest supports a wide variety of choices of who defines what in the user interface. This document focuses on what we call the "Informational Model".

The Information Model supplies the content structure associated with bonus material, but does not provide any navigation information. The Information Model is used when a pre-defined user interface expects specific information. Note that this model does not specify the user interface appearance beyond the inclusion of text and graphics that are used in the UI.

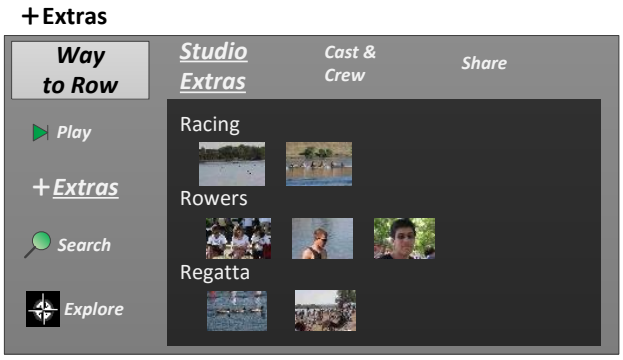
The information model assumes a user interface already exists and is supplying information to fill in the portion of the user interface associated with information from the Experience.

Consider the following example based on a hypothetical movie, "Way to Row". This example complies with Interactivity Profile 1 described later in this document.

The top-level screen is provided by the player. This screen has four options: *Play*, *Extras*, *Search* and *Explore*. The Profile Player implements *Search* and *Explore*, with no additional information from the Manifest, other than metadata.

The Play button leads to a screen like the following that plays the movie with additional material offered on the screen (perhaps when a button is pressed). The first screen shows Shorts and the second shows Galleries (after Galleries tab is selected). We call the data associated with these screens "In Movie".

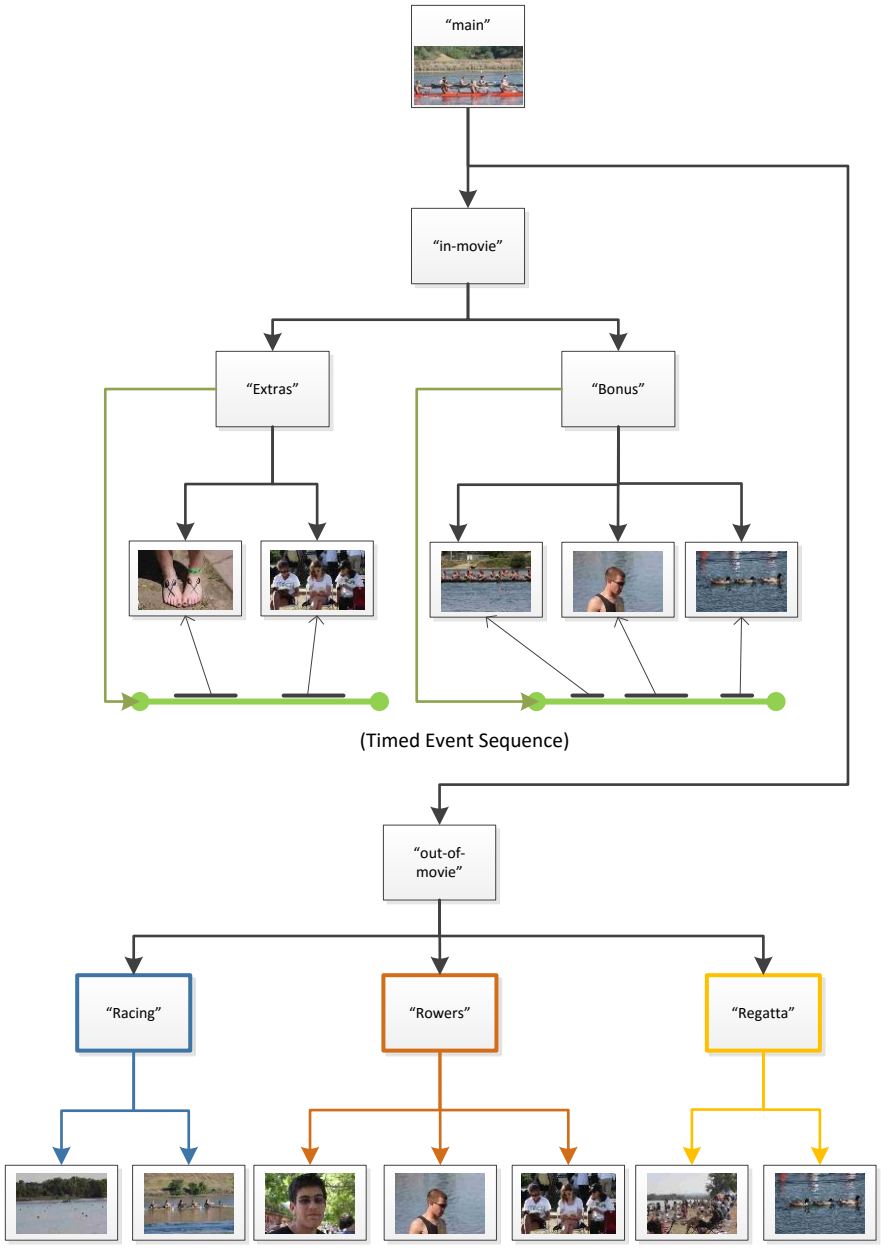
Back to the original screen, if the user selects Extras they get the following screen. We call the data associated with this screen "Out of Movie".



Notice that there are three tabs along the top, Studio Extras, Cast & Crew, and Share. The Studio Extras option is based on Media Manifest. The others are implemented by the player, perhaps using metadata and external data such as social network feeds.

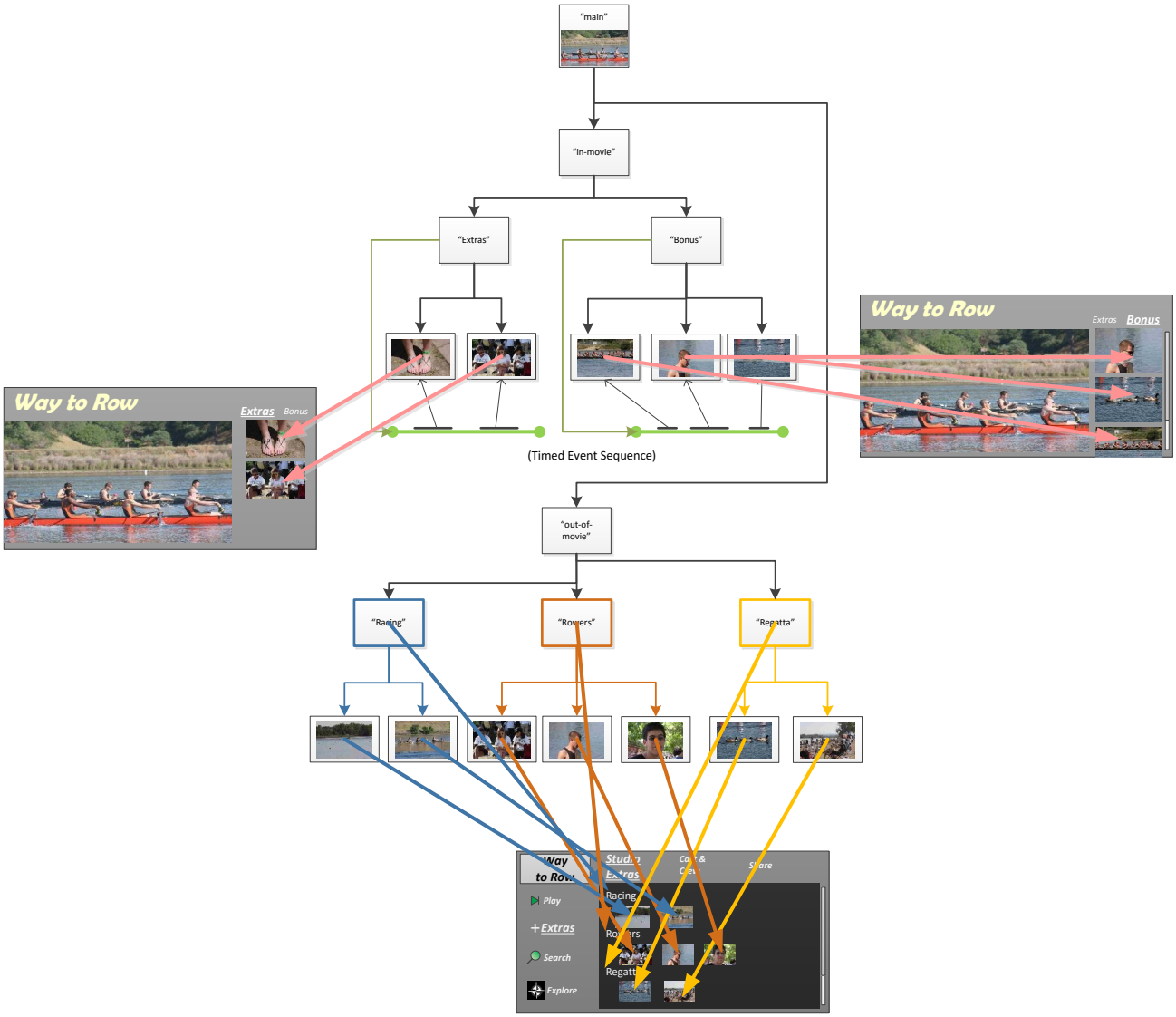
Information is fed from a Media Manifest as illustrated below. In this illustration, each box represents an Experience element. This example has a "main" Experience that references the film in an Audiovisual instance. It also has an "in-movie" (used during playback) Experience to describe the Play screens. And, it has an "out-of-movie" (used outside of playback) Experience that describes the extras on the "Studio Extras" tab.

Each in-movie child has a Timed Event Sequence associated with it. This information is used by the Player to synchronize the content with playback.



The trees under "in-movie" and "out-of-movie" drive the In Movie and Out of Movie screens illustrated above. All other screens are driven by the player application.

The following picture illustrates how all objects in the Manifest map onto the user interface.



5 MANIFEST ENCODING CONSTRAINTS

5.1 Metadata

This section defines which metadata to include.

The best references for metadata are Common Metadata [CM], Best Practices for Delivery [Delivery] Section 6, and Media Manifest Core [MMC] Section 4 (which references Media Entertainment Core [MEC]). These should be followed except where they conflict with specific recommendations in this document. For example, Audiovisual metadata (i.e., Audiovisual/ContentID) is not required as there is at most one Audiovisual instance in Experience element.

5.1.1 Metadata Source

Each Experience instance must include a ContentID element referencing metadata (i.e., ContentID is mandatory). The referenced metadata must be in the Inventory (i.e., Inventory/Metadata). The Metadata/Alias mechanism may be used.

Images should be handled in accordance with the recommendations in [Delivery], Section 4.2.3. In particular, LocalizedInfo/ArtReference should reference the Inventory via an Image ID. And, there should be a PictureGroupID in the Experience referencing a Picture Group that references all the metadata images. Any indirection to image locations should occur through the inventory.

It is recognized that certain implementations may prefer images to be directly referenced from ArtReference as a URL. This is allowable if both the author and player organizations agree.

Note: Using practices defined in [Delivery] allows interactivity implementers to take advantage of existing processes such as adding new localization updating media files.

5.1.2 Required Metadata

There are currently three defined uses for metadata

- Media Metadata – typical metadata use to describe content
- UI metadata for Information Model – text and graphical elements used in an Information Model-based UI

5.1.2.1 General Metadata Rules

Following are general metadata constraints:

- Required elements and attributes must be included.
    - WorkType must be correctly encoded as per [CM] and [Delivery].
    - RunLength should be 0 duration if not otherwise applicable.

- If there is more than one instance of LocalizedInfo, the default localization will be the first instance and have @localized='true'

5.1.2.2 Media Metadata

Basic Metadata should include the subset defined as part of the Media Entertainment Core (MEC) [MEC] definitions for Basic Metadata (Section 2.2.1). Additional metadata is allowed.

Metadata associated with each track in the Inventory should, at a minimum, include those data defined in MEC, [MEC] definitions for Digital Asset Metadata (Section 2.2.2). Note that the Inventory schema defines metadata using Common Metadata types, so there is a direct correspondence between what is specified in MEC and what is in the Inventory.

5.1.2.3 Metadata for Interactivity Model UI

The UI uses metadata as UI elements. Most relevant information is in LocalizedInfo. As this is user-facing, localizations should be provided for each language of interest.

The exact usage depends on the Profile information. The profile will specify where text is used, where images are used and constraints on those text and images (e.g., text length; and size and aspect ratio).

The following rules apply to top-level grouping nodes (i.e., those that are not presented to the user), such as "in-movie" and "out-of-movie" in the examples above:

- There is only one instance of LocalizedInfo
- That instance has TitleSort set to the name of the grouping category (e.g., "in-movie" or "out-of-movie").
- LocalizedInfo@language may contain any language code, and it is ignored.
- No other metadata is present unless it is a required element or attribute in the schema.

The following rules apply to nodes that contain user-visible information, such as a title (e.g., "Galleries"), an image, or both.

- An instance of LocalizedInfo should be included for each language supported for the title.
- LocalizedInfo/TitleDisplayUnlimited and TitleSort contains the user-visible name for that node. Note that even when an image is the intended UI element, text should still be provided for accessibility (text to speech).
- LocalizedInfo/ArtReference includes images associated with the node. Implementers note: Implementations should accept ImageID, PictureID, PictureGroupID and URL.

5.2 Applications

Applications are difficult because not all applications can be executed in all environments. The goal is to support a graceful degradation so that at least some functionality is available across all

platforms. In the worst case, it should be possible to omit the application without substantial negative impact on the UI.

5.2.1 Application Groups

The Application Group, documented in [Manifest], Section 7.1, provides the mechanism to signal the same application implemented for different platforms. All applications in the Application Group should offer the same function. Each application in an Application Group supports a particular usage (e.g., HTML vs. AppData vs. native application).

It is up to the Player to compare the Compatibility element to its own capabilities and choose the appropriate app.

5.2.2 Graceful Degradation

There are generally two ways to fully implement an application

- Natively – the Profile Player understand the application function and has the data necessary to implement that function
- HTML – The Profile Player supports HTML (e.g., it is a hybrid application), and there is an HTML application with the necessary data

Profile Players might support, one, both or neither. In addition to these options, it is possible to specify an image to display in lieu of the application. This is graceful degradation for some applications. For example, if a mapping application could not be implemented in HTML, it might be sufficient to show a map image with the point of interest highlighted.

The content author lists all possible applications by listing them as distinct InteractiveTrackReference instances. The content author can also specify which is preferred using the InteractiveTrackReference/@priority attribute (lower values are higher priority).

The Profile Player can use the information to provide the best possible experience. The Profile Player should play the application with the highest priority that it is capable of playing. The choice of which one goes with the content author. That is, if a Profile Player is capable of playing and application with higher priority it should do so.

If the application cannot be implemented and there would be no ‘hole’ in the UI (e.g., a menu list with no items) the application should be omitted. If a menu item has child elements that only consist of applications that cannot be played, that menu item is not shown. Authors should take care to avoid situations where this may occur.

For example, consider the case where there is native app (@priority=0, an HTML app (@priority=1) and an image (@priority=2). If the player can play the native app it does so. If it can’t but can play HTML it does so. If can neither support the native app or HTML it provides the image. If there was not an image provided and the player could not play the native app or HTML, it would ignore the app entirely (e.g., not display it as an option on the screen).

When an image is a sufficient replacement for the application, an Interactive element is created in the Inventory as follows:

- Type='Image'
- Encoding/RuntimeEnvironment='Default'
- PictureID references a Picture that contains the default images. Note that these can be localized—this is why PictureID is referenced instead of ImageID.

5.2.3 Application Reference and Delivery

Currently, the model for executing application is to follow an URL to HTML5/JavaScript.

If a platform supports specific application types, these can be included in the Manifest. However, delivery is outside the scope of this specification. Generally, these will be delivered in a manner similar to delivery of content.

Where application-specific data is required, we recommend using Manifest Application Data. This specification is not yet published, but if you are interested, please contact MovieLabs. This can support both built-in apps and HTML apps.