Cross-Platform Extras Application Data

CONTENTS

REVISION HISTORY

Version	Date	Description
1.0	July 19, 2016	Initial release. Schema was originally published as *draft* in July. It was *released* without changes on December 23, 2016.
1.1	December 23, 2016	Updated for Common Metadata 2.5 and Media Manifest 1.6.

1 INTRODUCTION

This document defines application support within Cross-Platform Extras (CPE). It defines data structures and their intended usage for both CPE-HTML and CPE-Manifest. It exists independently from those specifications to allow applications to evolve without impacting the core specs.

In this context, an application (or app) is any behavior that is part of an interactive experience that is not specifically defined by CPE-Manifest or CPE-HTML. Examples of applications are mapping, trivia, feeds and scripts. Support for these resides within Media Manifest, but additional information is required for the best possible support.

This document is part of the CPE family of specifications found at www.movielabs.com/cpe.

1.1 Overview

1.1.1 Technical Approach

This document builds on Media Manifest Metadata, providing application-specific information.

1.1.2 Extras Architecture

The Extras Menu architecture has the following data objects

- [TBS]

From these components an Extras Menu can be created.

1.2 Document Organization

This document is organized as follows:

1. Introduction—Provides background, scope and conventions
2. Data Model—Defines the data model and intended usage
3. Application Data Structure—Details of XML data structure and appropriate encoding and interpretation

1.3 Document Notation and Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119]. That is:

- "MUST", "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- "MUST NOT" or "SHALL NOT" means that the definition is an absolute prohibition of the specification.
- "SHOULD" or "RECOMMENDED" mean that there may be valid reasons to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- "SHOULD NOT" or "NOT RECOMMENDED" mean that there may be valid reasons when the particular behavior is acceptable, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- "MAY" or "OPTIONAL" mean the item is truly optional, however a preferred implementation may be specified for OPTIONAL features to improve interoperability.

Terms defined to have a specific meaning within this specification will be capitalized, e.g. "Track", and should be interpreted with their general meaning if not capitalized.

Normative key words are written in all caps, e.g. "SHALL"

1.3.1 XML Conventions

XML is used extensively in this document to describe data. It does not necessarily imply that actual data exchanged will be in XML. For example, JSON may be used equivalently.

This document uses tables to define XML structure. These tables may combine multiple elements and attributes in a single table. Although this does not align with schema structure, it is much more readable and hence easier to review and to implement.

Although the tables are less exact than XSD, the tables should not conflict with the schema. Such contradictions should be noted as errors and corrected.

1.3.1.1 Naming Conventions

This section describes naming conventions for Common Metadata XML attributes, element and other named entities. The conventions are as follows:

- Names use initial caps, as in InitialCaps.
- Elements begin with a capital letter, as in InitialCapitalElement.
- Attributes begin with a lowercase letter, as in initiaLowercaseAttribute.
- XML structures are formatted as Courier New, such as `md:rightstoken`
- Names of both simple and complex types are followed with "-type"

1.3.1.2 Structure of Element Table

Each section begins with an information introduction. For example, "The Bin Element describes the unique case information assigned to the notice."

This is followed by a table with the following structure.

The headings are

- Element—the name of the element.
- Attribute—the name of the attribute
- Definition—a descriptive definition. The definition may define conditions of usage or other constraints.
- Value—the format of the attribute or element. Value may be an XML type (e.g., "string") or a reference to another element description (e.g., "See Bar Element"). Annotations for limits or enumerations may be included (e.g.," int [0..100]" to indicate an XML xs:int type with an accepted range from 1 to 100 inclusively)
- Card—cardinality of the element. If blank, then it is 1. Other typical values are 0..1 (optional), 1..n and 0..n.

The first row of the table after the header is the element being defined. This is immediately followed by attributes of this element, if any. Subsequent rows are child elements and their attributes. All child elements (i.e., those that are direct descendants) are included in the table. Simple child elements may be fully defined here (e.g., "Title", " ", "Title of work", "xs:string"), or described fully elsewhere ("POC", " ", "Person to contact in case there is a problem", "md:ContactInfo-type"). In this example, if POC was to be defined by a complex type defined as md:ContactInfo-type. Attributes immediately follow the containing element.

Accompanying the table is as much normative explanation as appropriate to fully define the element, and potentially examples for clarity. Examples and other informative descriptive text may follow. XML examples are included toward the end of the document and the referenced web sites.

1.3.2 General Notes

All required elements and attributes must be included.

When enumerations are provided in the form 'enumeration', the quotation marks ('') should not be included.

The term "Device" refers to an entity playing the interactive material specified here. It may be a standalone physical device, such as a Blu-ray player, or it might be an application running on a general purpose computer, a table, phone or as part of another device. The term 'User' refers to the person using the Device.

1.4 Normative References

[CM]	Common Metadata, www.movielabs.com/md/md
[Manifest]	Common Metadata Media Manifest Metadata, www.movielabs.com/md/manifest
[CPE-Manifest]	Cross-Platform Extras, Manifest, www.movielabs.com/cpe/manifest
[CPE-HTML]	Cross-Platform Extras, HTML, www.movielabs.com/cpe/html
[CPE-BP]	Cross-Platform Extras Best Practices, www.movielabs.com/cpe/practices
[Ratings]	Common Ratings Metadata, www.movielabs.com/md/ratings
[RFC4646]	Philips, A, et al, *RFC 4646, Tags for Identifying Languages*, IETF, September, 2006. http://www.ietf.org/rfc/rfc4646.txt
[RFC4287]	IETF RFC 2460, The Atom Syndication Format, December 2005. http://tools.ietf.org/html/rfc4287
[RFC5023]	IETF RFC 5023, The Atom Publishing Protocol, October 2007, http://tools.ietf.org/html/rfc5023, as modified by Errata 1304 and 3207
[ISO639]	ISO 639-2 Registration Authority, Library of Congress. http://www.loc.gov/standards/iso639-2/
[ISO3166-1]	Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes, 2007.
[ISO3166-2]	ISO 3166-2:2007Codes for the representation of names of countries and their subdivisions -- Part 2: Country subdivision code
[ISO4217]	Currency shall be encoded using ISO 4217 Alphabetic Code. http://www.iso.org/iso/currency_codes_list-1
[ISO8601]	ISO 8601:2000 Second Edition, *Representation of dates and times, second edition*, 2000-12-15.
[TTML]	Timed Text Markup Language (TTML) 1.0, W3C Proposed Recommendation 14 September 2010, http://www.w3.org/TR/ttaf1-dfxp/

[KML]	Keyhole Markup Language, Version 2.2, Open Goespacial Consortium (OCG), http://www.opengeospatial.org/standards/kml
[KML-schema]	Keyhole Markup Language XML schema, Version 2.2, http://schemas.opengis.net/kml/

1.5 Informative References

[ManifestBPI]	Media Manifest Best Practices for Interactivity, www.movielabs.com/md/manifest
[AdID]	Ad-ID advertisement identifier, www.ad-id.org

2 APPLICATION DATA MODEL

2.1 Scope of usage

This document defines a format and syntax to be used when providing data to applications used to support a CPE experience. Application data can be used as part of CPE-Manifest and CPE-HTML or used independently.

This document defines XML as the exchange format as it well-defined and is more easily validated than other data languages (e.g., JSON). Prior to the application seeing the data we expect in many cases the XML will be processed into something more application friendly. Such processing is, however, outside the scope of this document. Nevertheless, the semantics in this document apply regardless of what form the data ultimately takes. That said, application-specific documentation is typically the principle document for authoring and interpreting application data.

2.2 Name-Value Pair

The data model is a compromise between full flexibility and well-defined objects. The model is extremely general and fully extensible, but certain commonly used objects are specifically defined.

The model for flexibility is name-value pairs (AKA key-value, attribute-value, etc.) App data is expressed as a nonempty set (1 or more) or name value pairs. Some names are included in a controlled vocabulary, although the spec allows addition non-standard names to be used. Values fall into one of the (many) types in the schema.

For example, general values could be expressed something like {{“Name”, “Craig”}, {“FavoriteTree”, “Elm”}}. Well-defined values would be something like, {{“Contentid”, “md:cid:eidr-S:8E01-E746-F1B1-E27A-836C-L”}}.

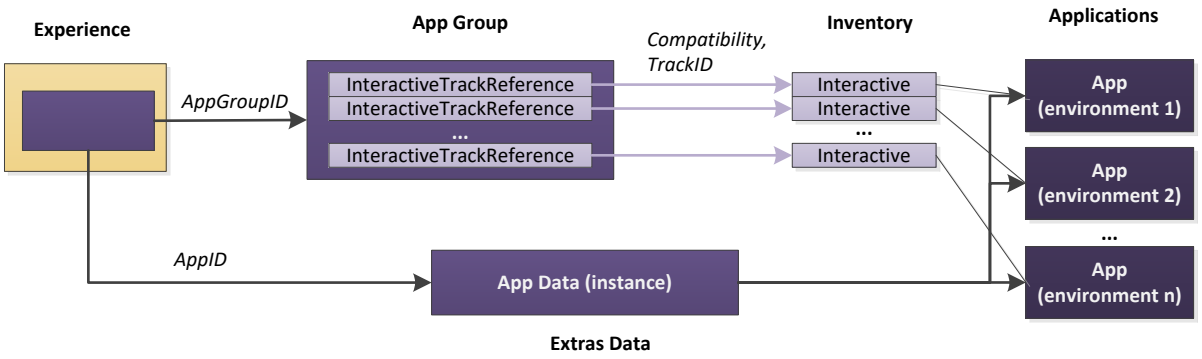
Types fall into the following categories

- Generic values – string, integer, monetary value, etc.
- Manifest Identifiers – ContentID, ExperienceID, etc.
- Special Use Identifiers – EIDR, Ad-ID, EAN/UPC, etc.
- Application-specific data – Location, Data feed, etc.

Names definitions need to be meaningful in the context of a specific application, so generally the names are defined as part of an application specification; generally, found in [CPE-BP].

2.3 Application Model in Manifest

The following diagram illustrates the relationships between application-related objects:



An Experience references an Application Group comprised of a set of application implementations that perform equivalent functions. As all apps do the same thing, they should all understand the same application data. Some app implementations may provide advanced and optional features lacking in a more basic implementation (e.g., a mapping app that supports altitude as well as just latitude and longitude). In these cases, all app implementations must have the same understanding of the subset of the common subset.

The Application Group, documented in [Manifest], Section 7.1, provides the mechanism to identify and access the same functionality implemented for different platforms. All applications in the Application Group should offer the same function. Each application in an Application Group supports a particular platform. For example, all applications in the Application Group might be mapping application; one for HTML, one for iOS, one for Android, one for PC, one for Mac, and so forth. This can be more granular if necessary (e.g., iOS 6-8, vs IOS 9).

The App Group references the Inventory that provides more details about the App, including where it can be found.

The App Group and Inventory is not necessarily specific to the Experience. For example, all mapping apps would be in the same App Group. If additional data is needed for the App Instance (i.e., the application behavior associated with that Experience), App Instance Data can be referenced by the Experience as @AppID. For example, while the mapping apps are in the App Group, the location for a map would be in the App Instance Data.

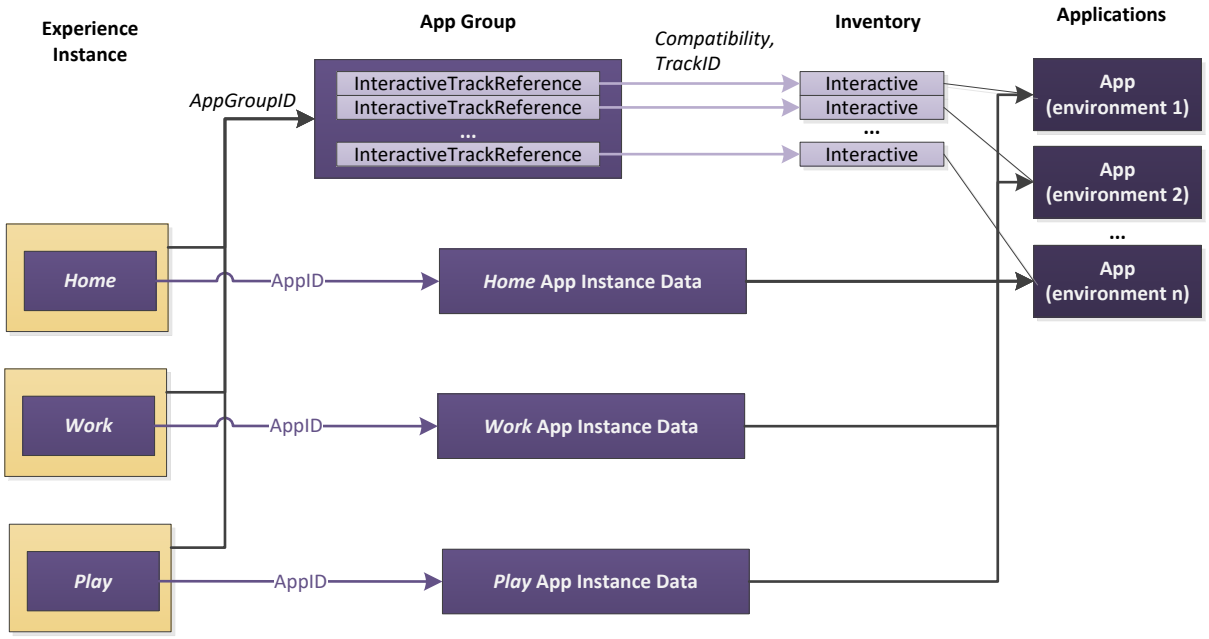
2.4 Using App Instance Data

As noted above, in Section 2.2, applications data is structured as name-value pairs. One or more name-value pairs can exist per application instance. Also, as previously noted, there may be more than one application implementation that understand the data (e.g., multiple

mapping applications). This section describes how to connect a specific Experience instance to the correct application data.

The @AppID attribute is the primary mechanism for identifying the correct data set to use with a given Experience. Data in Manifest Data's ManifestAppData applies to a given instance of an application in the Manifest's Experience/App when ManifestAppData/@AppID matches Experience/App/@AppID. All data in ManifestAppData applies to that application, although some implementations of that application (i.e., specific apps mentioned in an AppGroup) might use only selected items. The assumption is that application implementations will select only those names that apply.

For example, assume a Manifest that includes multiple Experience instances (Home, Work and Play), each referring to a specific location. In this scenario the @AppID in each Experience/App corresponds with the instance of App Data containing information about that location.



Note that the AppGroupID is not used when mapping an Experience to the appropriate App Data. Thus, each ManifestAppData/@AppID value must be unique within the context of a given ManifestAppDataSet.

3 APPLICATION DATA STRUCTURE

This specification assumes application data comes from the Media Manifest or from the Manifest Data object defined here. Data can come from other sources, although sources not defined here are likely to be less interoperable.

3.1 Manifest Data Set

The root object is ManifestAppDataSet that contains one or more instances of ManifestAppData. In most cases, only one instance of ManifestAppData will be included.

ManifestDataSetID-type is defined as md:id-type.

Element	Attribute	Definition	Value	Card.
ManifestAppDataSet-type				
	ManifestDataSetID	Unique identifier for this set of manifest data. This is used for version control when sending updates.	manifestdata:ManifestDataSetID-type	0..1
	updateNum	Version. Initial release should be 1. This is a value assigned by the metadata originator and should only be incremented if a new version of metadata is released. If absent, 1 is to be assumed.	xs:positiveInteger	0..1
ManifestID		ManifestID of Media Manifest that depends on this App Data Set.	md:id-type	0..n
ManifestAppData		Instance of a Media Manifest App Data Set	manifestdata:AppData-type	1..n

3.2 Manifest App Data

Manifest App Data is separately referenceable for purposes of tracking and updating. Consequently, it contains an identification attribute as well as @updateNum for versioning. It contains one or more instances of AppData; each instance corresponding with one application instance.

Element	Attribute	Definition	Value	Card.
AppData-type				
	AppID	ID used to reference data for this application instance. Corresponds with Experience/App/@AppID	md:id-type	
	updateNum	Version. Initial release should be 1. This is a value assigned by the metadata originator and should only be incremented if a new version of metadata is released. If absent, 1 is to be assumed.	xs:positiveInteger	0..1
Type		Type information for application	xs:string	0..1
SubType		Subtype information for application	xs:string	0..n
AppGroupID		AppGroupID associated with App Data (for validity checks)	manifestdata:AppGroupID-type	0..1
NVPair		Name Value Pair instance for each data objects	manifestdata:AppData-type	1..n

If ManifestAppDataSet/ManifestID is present, then AppID must be unique within all referenced Manifests. Otherwise, AppID must be globally unique.

3.3 Application Name-Value Data

AppNVPair-type contains various types of data that can be used by applications. Some elements are useful across various application types, and others are very specific to an individual application.

Generic objects are defined here. Specific objects are defined in subsequent sections of this document that are specific to given applications.

Element	Attribute	Definition	Value	Card.
AppNVPair-type				
	Name	Name part of name/value pair. The child elements in this element are the value part.	xs:string	

Type		Data type	xs:string	0..1
SubType		Data SubType	xs:string	0..n
AppGroupID		Reference to AppGroup that understands these data. This is used for consistency/validity checks, but is not otherwise required.	manifest:AppGroupID-type	0..1
The remainder of this object is a one or more instances of a xs:choice referencing elements of various types. That means that any number of these elements can be included in any order. Use is specific to the intended application. As the unbounded choice structure does not enforce order, these will be documented out of order. The last instance is an any##other object allowing any element from another namespace to be included.				

3.3.1 Generic Value Types

The following generic types are provided for the value part of the name-value pair.

Element	Attribute	Definition	Value
Part of AppNVPair-type			
Text		Any text string	xs:string
Integer		Any integer	xs:integer
Decimal		Any decimal number	xs:decimal
Duration		Time duration	xs:duration
URL		Any URI or URL	xs:anyURI
URLPostfix		A portion of a URL intended to be appended to a specific base URL. Typically, this will include some combination of path, query or fragment.	xs:string
Language		Language	xs:language
Time		Time	xs:time
YearDateTime		Year; year and date; or year, date and time, as per [CM]	md:YearDateOrTime-type

Money		Monetary value, optionally including currency, as per [CM]	md:Money-type
base64Binary		Any binary data or data best encoded using xs:base64Binary	xs:base64Binary
Location		Location (real or fictional), as per [Manifest]	manifest:EventLocation-type
Timecode		Any timecode	manifest:Timecode-type
Person		Person, character or group, as per [CM]	md:BasicMetadataPeople-type
TimePeriod		Time period, as per [Manifest]	manifest:EventPeriod-type
TimedEvent		Timed Event as per [Manifest]	manifest:TimedEvent-type

3.3.2 Manifest Identifiers

The following types are provided for Media Manifest and generic identifiers.

Element	**Attribute**	**Definition**	**Value**
Part of AppNVPair-type			
ExperienceID		Experience ID	manifest:ExperienceID-type
PlayableSequenceID		Playable Sequence ID	manifest:PlayableSequenceID-type
PresentationID		Presentation ID	manifest:PresentationID-type
ContentID		Basic Metadata Content ID	md:ContentID-type
PictureID		Picture ID	manifest:PictureID-type
TextGroupID		Text Group ID	manifest:TextGroupID-type
ALID		Logical Asset ID	md:LogicalAssetID-type
EIDR		EIDR	md:LogicalAssetID-type
OtherID		Any other identifier,	manifest:OtherID-type

OtherID should be used for identifiers not otherwise included in the schema. OtherID should not be used when the type of the ID in question is already provided as a distinct element definition in AppNVPair-type. If the ID corresponds with an identifier defined in Common Metadata [CM], Table 2-1 the value from the Scheme column should be used in OtherID/Namespace and the OtherID/Identifier should correspond with the "Expected value for <SSID>" column.

3.3.3 Special Use Identifiers

The following types are provided for commonly used non-Manifest identifiers. They are included to facilitate stronger type checking.

Element	Attribute	Definition	Value	Card.
Part of AppNVPair-type				
ProductID		Reference to a product that is well-defined in another namespace.	manifest:OtherID-type	
AdID		Ad-ID as defined in [ADID].	xs:string, pattern "a-zA-Z1-9][a-zA-Z0-9]{10}[hHdD]?"	
EANUPC		Numeric value associated with an International Article Number (EAN) or Universal Product Code (UPC)	xs:string	
	format	How to interpret the digits in the EANUPC value. See below.	xs:string	0..1

EANUPC/@format should be encoded as follows:

- 'UPC' – refers to 12-digit Universal Product Code, also referred to as GTIN-12
- 'GTIN-13' – refers to the values in a 13-digit EAN

3.4 Application-Specific Data

Element	Attribute	Definition	Value	Card.
Part of AppNVPair-type				
Gallery		Additional information to describe a gallery.	manifestdata:AppDataGallery-type	
LocationSet		A location point that describes a location, or a collection of location points that defines an area location.	manifestdata:AppDataLocation-type	
SelectTrack		Information about which track a user can select.	manifestdata:AppDataTrackSelection-type	
DataFeedSet		Description of data feeds.	manifestdata:AppDataFeed-type	
TimedEventSequence		Timed Events	manifest:TimedEventSequence-type	
AcquireAsset		Information about how to purchase, rent or otherwise acquire an asset associated with the CPE experience.	manifestdata:AppDataAcquireAsset-type	
KML		Keyhole Markup Language (KML) [KML]	manifestdata:KMLAppDataKML-type	
any##other		Additional objects	xs:any ##other namespace	

3.4.1 Gallery Data

Element	Attribute	Definition	Value	Card.
AppDataGallery-type				
	GalleryID	Gallery ID referencing Gallery that will be displayed.	manifest:GalleryID-type	

AutoNextSlideTime		Time each Picture dwells on screen before switching to next Picture. If '0', images should be switched manually. If absent, Device may select its own time, or choose not to switch images automatically.	xs:duration	0..1
Loop		Should images be displayed in a loop? That is, should first image be displayed after last image? If absent or 'false' images are not looped. If 'true' images are looped.	xs:boolean	0..1

3.4.2 GIS Data

3.4.2.1 AppDataLocation-type

The Location/mapping application provides a geographic display. It may be real or fictional. It may be earthbound or elsewhere.

Element	Attribute	Definition	Value	Card.
NodeLocation-type				
Location		A location. One instance is included for each location associated with this context	manifest:EventLocation-type	1..n
	icon	Reference to image to be used to mark this location on the map.	manifest:ImageID-type	0..1
MapImageID		An image that can be used as map.	manifest:ImageID-type	0..1

If MapImageID is included, the Location should include OtherCoordinates encoded as follows

- system='image'
- Coordinate are as follows:
  - @label='x', and Coordinate value is the offset in number of pixels horizontally from upper left corner of image.
  - @label='y', and Coordinate value is the offset in number of pixels vertically from upper left corner of image.

Note that the inclusion of (x,y) OtherCoordinate instances does not preclude the inclusion of additional OtherCoordinate instances or an EarthCoordinate instance.

3.4.2.2 AppDataKML-type

This is defined a type to allow a document to be created both with and without validation. Validation requires editing the schema, although the necessary data are in comments. Depending on what is commented, the object is defined as one of the two. With schema editing for maximum validation:

Element	Attribute	Definition	Value	Card.
AppDataKML-type				
kml		KML document as per [KML]. This form requires schema editing.	kml:kml	

and, without schema editing:

Element	Attribute	Definition	Value	Card.
AppDataKML-type				
kml		KML document as per [KML]. Document should still use a kml:kml object.	xs:any ##other	

3.4.3 Track Selection Data

Track selection is the process of selecting the desired video, audio and subtitle tracks for playback. The track selection model is extended to include timeline 'tracks', allowing the user to select which timeline objects are presented and which are filtered.

Audio, video and subtitle track selection are 'radio buttons' in the sense that only one of each can be selected.

The process for selecting the default video, audio and subtitle tracks is defined in [Manifest], Annex A. Players should implement this algorithm and present the default track as the pre-selected option.

Element	Attribute	Definition	Value	Card.
AppDataTrackSelection-type				
PresentationID		Presentation for TrackID	manifest:PresentationID	
SelectVideoTrack		Are video tracks offered for selection?	xs:boolean	
SelectSubtitleTrack		Are subtitle tracks offered for selection?	xs:boolean	
SelectAudioTrack		Are audio tracks offered for selection?	xs:boolean	

3.4.4 Feed Data

Data feeds are difficult to handle given that there are so many specialized feeds that require particular attention. The goal of this model is to provide a simple interface that all players can implement while still providing any additional information that would be required for specific feeds. For example, a generic player could capture Twitter text over a simple feed while a more advanced player could implement the Twitter APIs and provide a full experience.

The Data Feed model defines the source and information needed to obtain the information. The specification defines one (or possibly more models). It is assumed that a player will communicate with an intermediary that can convert a proprietary feed to the standards format. Additional information is provided that allows a player to access a proprietary feed directly.

A Feed Set consists of one or more sources for the same data. The player should select the richest feed it is capable of playing. For example, if it has the capability to provide a full experience using a proprietary feed, it should use that feed over a generic feed.

AppDataDataFeedSet-type is a grouping entity that allows equivalent feeds to be listed.

Element	Attribute	Definition	Value	Card.
AppDataFeedSet-type				
Generic		Generic Feed (i.e., not proprietary to a given service).	manifestdata:AppDataFeedGeneric-type	1..n (choice)
Proprietary		Proprietary feed	manifestdata:AppDataFeedProprietary-type	

3.4.4.1 Generic Feed

The default feed is a feed defined by this specification. It is based on Atom [RFC4287] and [5023] (a derivative of RSS).

Atom usage in conformance with the Atom Publishing Protocol as defined in [RFC5023]. Players use the GET form of the Atom Publishing Protocol. POST, PUT and DELETE forms are be used.

At Atom feed is obtained by performing an HTTP GET to the URL in SourceURL. An Atom data feed in the form of an atom:feed element constrained as follow:

Element	Usage
author	Service providing the data; that is, the service providing the 'default' feed. The original service is identified by the feedType.
title	Title of feed. This not intended for presentation to the user as only one instance is allowed and this can therefore not be internationalized.
link	Link to this feed
id	As appropriate.
updated	Date and time when feed was updated
entry	One entry for each resource
entry/title	Title of resource. As appropriate.
entry/link	Link with href attribute referring to resource (see below)
entry/id	ID for Resource. As appropriate.
entry/updated	Date and time resource was created or updated

Data is obtained by performing an HTTP GET to the value in entry/link element of the Atom feed for the desired resource.

Data returned may use the following format (other formats can be acceptable)

Element	Attribute	Definition	Value	Card.
AtomFeedData-type				
FeedImageLocation		Location of image for feed (i.e., corporation)	xs:anyURI	0..1
FeedImageID		Manifest Image ID of image for entity that posted content.	manifest:imageID-type	0..1

PostingImageLocation		Location of image for user posting	xs:anyURI	0..1
PostingName		Name of user posting	xs:string	0..1
WhenPosted		When resource was posted. Can be resolution of year; year and date; or year, date and time	md:YearDateOrTime-type	0..1
Title		Title of post	xs:string	0..1
Body		Body of post. This can include links.	xs:string	
BodyImageLocation		Image associated with post	xs:anyURI	0..1

3.4.4.2 Proprietary Feed

A Proprietary Feed is a feed from a proprietary service. There are often SDKs provided to process data from proprietary feeds.

Element	**Attribute**	**Definition**	**Value**	**Card.**
AppDataFeedProprietary-type				
	category	Category of feed (see below)	xs:string	
	feedType	An identifier for the feed source type (see below)	xs:string	
	feedSubType	Any additional informat required to distinguish between feeds of a given feedType. This could include versioning information.	xs:string	0..1
SourceURL		URL where feed can be obtained	xs:anyURI	0..1
QueryObject		Information that must be posted to the feed to obtain the required information.	xs:string	0..1

category and feedType are each encoded using a unique identifying value. The following values should be used when the mentioned category and feed is used.

- category='generic' – Generic feed as defined in Section 3.4.4.1
    - feedType is the same value as it would be under a different category

- category='social'
  - 'twitter'
  - 'facebook'
  - 'instagram'
  - 'snapchat'
  - 'vine'
  - 'pinterest'
  - 'googleplus'
  - 'tumblr'
  - 'flikr'
- category= 'news'
  - 'rotten' – Rotten Tomatoes
  - 'ew' – Entertainment Weekly
  - 'nyt' – New York Times
  - 'time' – Time Magazine

3.4.5 Asset Acquisition Data

In order to acquire an asset, the information contained in NodeAcquireAsset-type passed to the retailer's purchasing function:

Element	Attribute	Definition	Value	Card.
AppDataAcquireAsset-type				
ALID		Logical Asset ID associated with title. Corresponds with ALID in Avails.	md:AssetLogicalID-type	
RequestType		Type of acquisition request (e.g., buy or rent).	xs:string	
FormatProfile		The media profile associated with the title. This corresponds with FormatProfile in [Avails].	xs:string	0..1

PurchaseURL		Link to site where content can be purchased. If there are multiple purchasing options, multiple instances can be included.	xs:anyURI	0..n

RequestType is encoded as follows:

- 'Buy' – Indicates purchase
- 'Rent' – Indicates rental
- 'Free' – indicates a free acquisition
- 'Acquire' – Indicates an acquisition that does not involve one of the above. This is used when the Retailer presents the options to the consumer.