Cross Platform Extras Interface

CONTENTS

NOTICES

This work is licensed under a Creative Commons Attribution 3.0 Unported License.

NOTE: No effort is being made by the Motion Picture Laboratories to in any way obligate any market participant to adhere to this specification. Whether to adopt this specification in whole or in part is left entirely to the individual discretion of individual market participants, using their own independent business judgment. Moreover, Motion Picture Laboratories disclaims any warranty or representation as to the suitability of this specification for any purpose, and any liability for any damages or other harm you may incur as a result of subscribing to this specification.

REVISION HISTORY

Version	Date	Description
V1.0	June 2, 2015	Initial posted version

1 INTRODUCTION

This document defines an architectural framework and API that is will support the deployment of content packages that enhance the user's viewing experience. Instead of just watching the movie, the user will be provided with a broad set of features that makes it a fully immersive experience. The goal of this document is to facilitate the coordinated efforts of both content producers and content retailers/distributors in the creation of these packages. To that end an API is defined, along with the supporting contextual material to allow interested parties to create, integrate, and deploy compliant components.

1.1 Scope

The API defined in this document is intended to facilitate the coordinated efforts of both content producers and content retailers/distributors in the creation of interactive viewing experiences. This document 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**). Direct interactions of either a Framework or a Package with 3rd-party components, such as operating systems, social networks, or back-end services, are outside the scope of this document.

The anticipated initial implementations of this specification will be HTML/ECMAScript (JavaScript). Although HTML5 is not fully available, the HTML5 features available across the most popular browsers will likely be used. Given this focus, we provide HTML/ECMAScript examples. As appropriate, other examples may be provided in the future.

1.2 Document Organization

This document is organized into three parts. The first part consists of introductory and overview material:

Section 1. Introduction - background, scope and conventions

Section 2. Primary Components - identification of top-level components and actors

Section 3. API – Technical overview of the API including its design patterns and organization

Sections 4 through 9 contain the normative specification of the Cross-Platforms Extras API organized into functional groupings. These are:

Section 4. Package Management

Section 5. Content Access

Section 6. Account Access

Section 7. Player Interaction

Section 8. Social Networking

Section 9. Enhancements

The remaining sections contain informative material intended to assist developers in understanding and implementing the API:

Annex A. Implementation Guidance

Annex B. Adaptation to Specific Viewing Environments

Annex C. Examples

1.3 Relationship to other Specifications

This specification is designed to be compatible with specifications anticipated to be used on conjunction with CPE. We have paid particular attention to the following

- Media Manifest – The Media Manifest provides a structure for describing content, including additional video, galleries, and other assets and resources. The CPE proof of concept has demonstrated that Media Manifest can be used as an integral part of defining an interactive experience.
- Common Metadata – MovieLabs Common Metadata provides standard encodings for many metadata objects.
- Common Ratings – Common Metadata Ratings defines standard encodings for ratings worldwide. It also provides information on ratings systems that can be used to construct parental control systems.

1.4 Document Notation and Conventions

As a general guideline, 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”.

Normative requirements need not use the formal language above.

1.4.1 Conventions

This API specifies interfaces intended to provide functionality in several areas, including package management, content access, media playback, and social networking. Developers may, if they choose, implement either enhancements to the existing capabilities or additional features outside the scope of this API. If, however, developers choose to do so, it must be done in a manner that does not conflict with the interfaces and state behaviors specified by this API.

1.4.2 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.

UTF-8 [RFC3629] encoding shall be used when ISO/IEC 10646 (Universal Character Set) encoding is required.

1.5 Normative References

[CSS]	Cascading Style Sheets Level 2 Revision 1, B. Bos, T. Çelik, I. Hickson, H. Lie. W3C., http://www.w3.org/TR/CSS2/
[ECMA-262]	ECMAScript Language Specification, Edition 5.1, June 2011, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
[EIDR-2.0]	EIDR System Version 2.0 Data Fields Reference, Feb 6, 2014, http://eidr.org/documents/EIDR_2.0_Data_Fields.pdf
[HTML5]	A vocabulary and associated APIs for HTML and XHTML, Candidate Recommendation 04-Feb-2014, R. Berjon, S. Faulkner, T Leithead, E. D. Navara, S Pfeifer, I Hickson, http://www.w3.org/TR/2014/CR-html5-20140204/
[JSON]	RFC-4627: The application/json Media Type for JavaScript Object Notation (JSON), July 2006, D. Crokford, http://www.ietf.org/rfc/rfc4627.txt
[XML]	Extensible Markup Language, T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau. W3C., http://www.w3.org/TR/xml/

1.6 Informative References

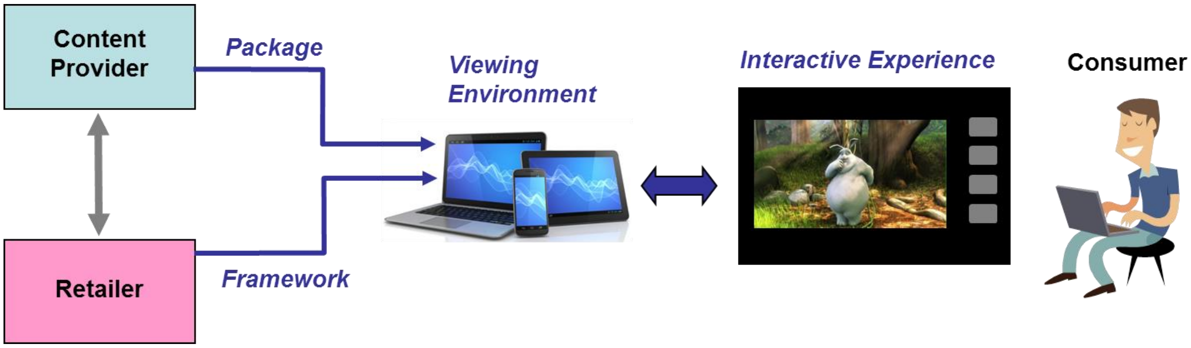
[TR-META-XTRA]	Common Extras Menu Metadata, v0.8e, March 2, 2015 http://www.movielabs.com/extras
[TR-META-MMM]	Common Metadata Media Manifest, v1.4, June 12, 2015. http://www.movielabs.com/md/manifest
[TR-META-CM]	Common Metadata, TR-META-CM, v2.3c, July 1, 2015, http://www.movielabs.com/md/md
[TR-META-CR]	Common Metadata Ratings, http://www.movielabs.com/md/ratings/

1.7 Best Practices for Maximum Compatibility

This API specifies interfaces intended to provide functionality in several areas, including package management, content access, media playback, and social networking. Developers may, if they choose, implement either enhancements to the existing capabilities or additional features outside the scope of this API. If, however, developers choose to do so, it must be done in a manner that does not conflict with the interfaces and state behaviors specified by this API.

2 PRIMARY COMPONENTS

The CPE is defined in terms of Content Providers who create Packages, Retailers who create Frameworks, and Viewing Environments that execute the Package and Framework to create an Interactive Experience for a Consumer. This is illustrated below.



More formally:

- **Consumer**: the individual(s) viewing the content in their home via an on-demand and interactive platform (e.g., streaming to a tablet device).
- **Content Provider**: the organization that owns the content (e.g., a movie studio).
- **Retailer/Distributor**: the organization that is responsible for making the content available to consumers and the collection of any associated fees (e.g., streaming services or on-line retailers).

CPE defines APIs in terms of three conceptual components:

- **Viewing Environment**: consumer's hardware platform and operating system (e.g., Android tablet, iPhone, Windows 8 PC, etc.).
- **Framework**: software that allows the consumer to use their selected viewing environment to view, and interact with, content. The framework may include components running within the consumer's viewing environment as well as components residing on servers or cloud-based platforms.
- **Package**: set of content files and supporting control software that defines the state behavior of the interactions. The package content files will consist of both primary content (i.e., the movie that is the focus of the consumer's attention) and the supporting content (e.g., behind-the-scenes commentary, deleted scenes and interviews).

3 API

The method signatures and data structures that comprise the Cross-Platform Extras API are defined in Sections 4 thru 9 of this document. To facilitate an understanding of the behavior and intended usage of these specifications, an overview of the API and the underlying design principals is first presented in this section of the documentation.

3.1 Overview

The Cross-Platform Extras API is organized into functional groups

- **Package Management:** addresses both management of the life-cycle of the package, and access by the package to information regarding the Viewing Environment.
- **Content Access:** functions to identify and/or acquire the rights and entitlements to access content as well as the ability to transfer (i.e., download) the content from its current location.
- **Account Access:** provides a package the ability to establish the identity of the consumer via a sign-in process and then obtain or modify information specific to the consumer (e.g., preferences, wish lists).
- **Player Interaction:** deals with the actual playback of a content file or stream. Methods are provided for obtaining a media player and then directing its operations (i.e., play, pause, trickplay, etc.).
- **Social Networking:** deals with the use of social media accounts to interact with a larger community via posts, likes, and similar mechanisms.
- **Enhancements:** functionality that is considered optional. The API therefore provides the package with the ability to query the framework to determine which specific capabilities are provided.

Each of the following sections provides details for one these API groups.

3.2 API Applicability

The following table indicates which APIs are required for implementation.

Mandatory (“M”) APIs that must be implemented by the Framework or Package as indicated.

Optional (“O”) APIs are strongly recommended for the Framework to ensure all Package features can be used with that Framework. For the Package, optional APIs are implemented as necessary to support features of the Package’s experience. Framework implementers may decide bilaterally with Package implementers that certain optional APIs must be implemented.

Some APIs are not applicable (‘N/A’) to the Package and are marked accordingly.

An interface is provided in the Package Management Group to determine which API Subgroups are implemented.

Group	Subgroup	Framework API	Framework Events	Package
Package Management	Lifecycle	M	N/A	M
	Connectivity	O	N/A	O
	Environment	O	N/A	O
Content Access	Availability	M	M	N/A
	Access Event	M	N/A	M
	Download	O	N/A [note 1]	N/A
Account Access	Basic	M	M	N/A
	Account Event	M	N/A	M
Player Interaction	Lifecycle	M	M	N/A
	Basic	M	M	N/A
	Trickplay	O	M [note 2]	N/A
	Controls	O	M [note 2]	N/A
	Sound	O	M [note 2]	N/A
	Player Event	M	N/A	M
	Geometry	M	M	N/A
Social Networking	Sharing	O	O [note 3]	N/A
	Social Event	M	N/A	M
Enhancements	Wishlists	O	N/A	N/A
	Bookmarks	O	N/A	N/A
	Package History	O	N/A	N/A

Notes:

1. Download events are signaled using the StatusDescriptor completion codes. See Section 4.4.2
2. Frameworks must be prepared to notify a Package of these events when they are the result of the Consumer using the Player’s internal control UI. Event support is

therefore MANDATORY even for Frameworks that do not expose this portion of the API to the Package.

3. Frameworks that implement the Sharing subgroup must also support the related events.

3.3 API Design Patterns

The Cross-Platform Extras API has been developed with the goal of supporting the operational concepts discussed in Section A.1. To that end, several software design patterns have been adopted to ensure that these goals will be met.

3.3.1 Zero-Argument Constructors

The API is based on the use of zero-argument constructors to facilitate a plug-in design pattern. Thus, the API includes ‘setter’ methods for all required operational parameters. It also allows for state behavior in which a receiving construct may reject a request due to having insufficient state information to execute (i.e., initialization is incomplete).

3.3.2 Event Notification via Registered Listeners

Notification of asynchronous events is handled via notifications sent to registered listeners. There are two situations in which this takes place: as a result of an unexpected event or upon the completion of a previously requested task.

3.3.3 Completion of Asynchronous Services

When asynchronous tasks terminate, either successfully or not, notification of final status as well as access to resulting data structures (e.g., a downloaded file) is handled via a notification of the appropriate listener instance.

A service request that requires activity that is potentially ‘heavy-weight’ or subject to network latencies should be handled asynchronously. That is, if a service implements a function that could take a long time to return, for example, checking consumer credentials with a central server or downloading media files, it should return promptly and perform the bulk of the processing asynchronously.

Return codes from these services are only indications that service provider is able to *attempt* to satisfy the request. A successful response means that the request parameters provide sufficient information for an attempt to be made and that the service provider has no reason to believe that the required infrastructure (e.g., network connectivity, storage space) is not available.

One should assume that a returned ‘SUCCESS’ from a heavy-weight method only means that the caller provided the info necessary to validate the request and queue it up for asynchronous handling later. It may subsequently fail. For example, the service may validate a URL is formed correctly and return success, but later fail if that URL addresses an unreachable service.

3.3.4 Error Handling

Error handling is designed to allow a package to gracefully recover from errors. This requires consistent and informative error returns. Error and status messages provide:

- enumeration of error, event, and status codes
- support for logging of diagnostic information
- support for display of user-friendly messages to the consumer

Components receiving an error notification are to perform any heavyweight recovery procedures asynchronously. Error recovery should not be performed on the notification thread. so as to isolate and protect all components and subsystems from failures in other components.

3.3.5 Player State Behavior

The choice of media player is the responsibility of the Retailer. A player and/or package will provide UI controls allowing user to directly manage the media playback. The media player is, therefore, defined in terms of two distinct state machines:

- lifecycle (i.e., create, terminate, and destroy)
- playback (i.e., start, pause, fast-forward, stop, etc.)

See Section 7, *Player Interaction* for further details.

Note that it is anticipated that in the future players will be implemented using HTML5, Media Source Extensions and Encrypted Media Extensions.

3.4 Interfaces

The CPE interface treats the package and the framework as asynchronous state-machines. Either entity may therefore be the initiator of a method call. The following material is grouped, therefore, first by functional area (i.e., content access, player management, etc.) and then by which entity is the initiator. Interfaces are, therefore, paired (e.g., a framework must provide an object instantiating the `Framework` interface while a package must provide an object instantiating the `Package` interface).

Any constants (e.g., status code or flags) that are passed across the interfaces will also be defined in the appropriate subsection.

3.5 Content Identification

Content is identified by a string called `contentID`. It is essential that what is referred to by contentID is agreed upon by the Framework and Package, and that contentID is sufficiently precise that the correct content is referred to.

3.5.1 ContentID Format

ContentID can be any format as long as it is agreed up on by the Package Implementer and the Framework Implementer.

Content ID format defined in Common Metadata [TR-META-CM], Section 2 SHALL be used. This format is: "`md:contentID:`"<scheme>"`:`"<SSID>

<scheme> and <SSID> are defined in [TR-META-CM].

Note that this does not constrain the use of particular identifier schemes (e.g., EIDR or UUID). Any identifier can be represented in this format.

3.5.2 ContentID Consistency

A goal of CPE is to create a single Package that runs across multiple Frameworks. If content identifiers must be unique for each Framework, then a Package is not fully portable. To achieve portability, it is essential that identifiers be used uniformly.

Within CPE `contentID` always refers to content that will be played by the Framework. The Package implementer and Framework implementers must agree upon which ID is used.

The same ID (`contentID`) must be used for availability/entitlement, playback and download.

Identifiers used SHALL be as defined by the Content Provider. That is, if a studio provides a Retailer with a particular identifier, that identifier is used as `contentID`.

It is strongly recommended that EIDR identifiers be used as specified in [EIDR-2.0]. The use of a standard identifier has substantial benefit to interoperability. A specific benefit is the ability to accept identifiers that are not an exact match for a given encoding.

3.5.2.1 Note on Consistency (informative)

The key concept in identifiers is that the Framework (Retailer) and Package (Studio) agree. This means gives the implementers some flexibility in using identifiers in a manner that would otherwise be considered misuse. This is intentional because it allows parties to work with existing identifiers rather than force them into a particular identifier model.

In some case `contentID` might be very specific; and in other cases more general. It does not matter as long as `contentID` can be used across APIs. For example, if the entitlement is "Movie A, Director's Cut" and a `getAvailability`() call indicates it's available, then the Framework can perform a `createPlayer`() with that same `contentID` and expect it to play.

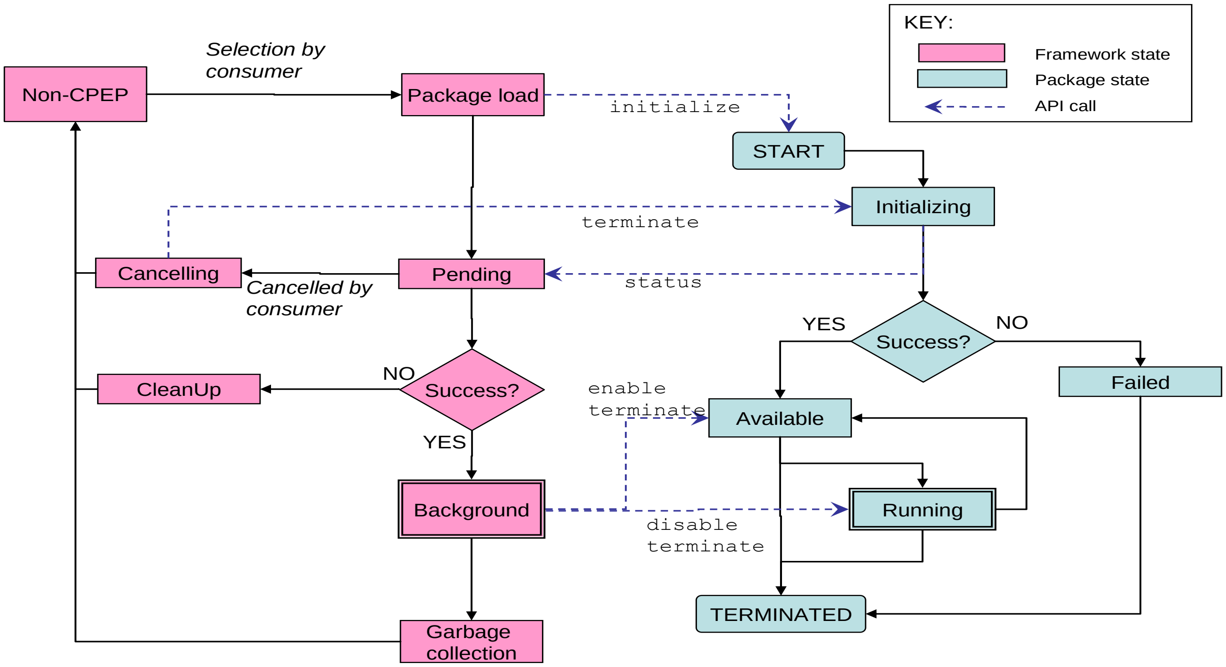
4 PACKAGE MANAGEMENT API GROUP

The Package Management API Group addresses both management of the life-cycle of the package, and access by the package to information regarding the Viewing Environment.

4.1 Overview

4.1.1 Package Lifecycle

The Package life-cycle is shown in the diagram below. Only those method calls that initiate a state transition are shown. Note that Non-CPEP refers to the state prior to the CPE package being invoked.



Key points to note are:

- The Framework may at any time terminate a Package.
- The condition when a consumer is watching a movie is to have a Package in the `Running` state and the Framework in the `Background` state.
- The framework may temporarily disable a package, thereby taking back control of the UI, then re-enable to package. This is NOT the same as pausing/resuming playback in that the later situation does not change the UI view.

4.1.2 Properties of the Viewing Environment

A package will require information about the viewing environment in which it is operating. While a Package has the option of directly accessing the viewing environment via native methods, it may also obtain some information about the viewing environment from the Framework.

Environmental properties may be regarded as falling into one of two categories: *static* and *dynamic*. The static properties accessible via the Framework and this API are referred to as the Environment Descriptor. This structure contains information regarding the static properties of a consumer's hardware platform and operating system such as screen size, OS type and version, supported media formats, and other characteristics not expected to change during normal operations. Properties are specified in terms of key-value pairs (KVP).

Dynamic properties (e.g., current battery status, available storage space) are not considered to be part of the environment's description. Should a Package desire this type of information, or it will need to make use of mechanisms outside the scope of this API.

4.1.3 Structure and Subgroups

The Package Management API is divided into the following subgroups.

Subgroup Summary	
Lifecycle	Provides life-cycle management of a package by the Framework. Allows the Framework to notify the Package of changes in the viewing environment.
Connectivity	Methods for a Package to obtain information regarding the Viewing Environment's networking capabilities
Environment	Methods for a Package to obtain Viewing Environment information from the Framework

The following data structures are used to support the exchange of relevant information between the Package and Framework.

Package Management Data Structures	
`ConnectivityState`	Provides information regarding the current state of network connectivity
`StatusDescriptor`	Used to exchange status and error notifications

4.2 Framework Interface

4.2.1 Lifecycle Subgroup

Method Summary	
`status`	Signals to the framework a change in the status of the package or the results of a requested operation.

4.2.1.1 getSupportedAPIs()

Returns to the caller the set of API subgroups that are supported by the Framework. The supported groups are identified by codes:

Group	Subgroup	Code
Package Management	Lifecycle	GRP_PM_LC
	Connectivity	GRP_PM_CONN
	Environment	GRP_PM_ENV
Content Access	Availability	GRP_CA_AV
	Access Event	GRP_CA_EVT
	Download	GRP_CA_DLD
Account Access	Basic	GRP_AA_B
	Account Event	GRP_AA_EVT
Player Interaction	Lifecycle	GRP_PI_LC
	Basic	GRP_PI_B
	Trickplay	GRP_PI_TP
	Controls	GRP_PI_CTRL
	Sound	GRP_PI_SND
	Player Event	GRP_PI_EVT
	Geometry	GRP_PI_GEOM

Social Networking	Sharing	GRP_SN_SHARE
	Social Event	GRP_SN_EVT
Enhancements	Wishlists	GRP_EN_WISH
	Bookmarks	GRP_EN_BKM
	Package History	GRP_EN_PH

Usage

```
    getSupportedAPIs ()
```

Parameters: `none`:

Returns:

`code[]` : integer array of all applicable availability codes

ECMAScript Example

4.2.1.2 status( )

Signals to the framework a change in the status of the package or the results of a requested operation.

Usage

```
    status(StatusDescriptor status)
```

Parameters:

`status`: a `StatusDescriptor` instance

Returns: none

ECMAScript Example

4.2.2 Connectivity Subgroup

Method Summary	
`getConnectivityState`	Requests information regarding the current network connection

4.2.2.1 getConnectivityState()

Requests information regarding the current network connection. Depending on the viewing environment, some or all of the requested information may be unavailable.

Usage

```
getConnectivityState()
```

Parameters: none

Returns:

`state`: instance of the `ConnectivityState` data structure

ECMAScript Example

4.2.3 Environment Subgroup

Method Summary	
`getEnvironmentDesc`	Request for information regarding the static properties of a viewing environment (i.e., screen size, OS type and version, supported media formats, etc.).

4.2.3.1 getEnvironmentDesc()

Request for information regarding the static properties of a viewing environment (i.e., screen size, OS type and version, supported media formats, etc.). Properties are specified in terms of key-value pairs (KVP).

Usage

```
getEnvironmentDesc()
```

Parameters: none

Returns:

Set of key-value pairs – The keys and values of these key-value pairs is not currently defined in the specification as this API is provided for general flexibility. At some point, the specification may include certain controlled vocabulary.

ECMAScript Example

4.3 Package Interface

These methods allow the Framework to notify the Package of changes in the viewing environment and to manage the life-cycle of a package.

4.3.1 Lifecycle Subgroup

These methods provided for life-cycle management of a package by the Retailer's framework.

Method Summary

`initialize`	Instructs the Package to initialize itself
`enable`	Informs the Package it has control of the user interface and is now visible to the consumer.
`disable`	Informs the Package that control of the user interface is reverting to the Framework and is no longer visible to the consumer
`terminate`	Instructs the Package to immediately clean up as the Framework will be taking back control of the user interface and will terminate the Package.
`getState`	Requests the current state of the package.
`getExperienceId`	Requests the *Experience* implemented by the package

4.3.1.1 initialize()

Instructs the package to initialize itself.

The initialize() function will be invoked by the framework when it determines that the package is in the `LOADED` state. The mechanism by which this is determined will be specific to the implementation language. Refer to Annex B for further details.

Initialization is completed when the package is in a state allowing the control of the UI to be passed to it (i.e., it is in the `Available` state). The nature of the operations that are carried out during initialization may vary from package to package and may take some time if remote resources are required. The package is, therefore, allowed to complete the initialization process asynchronously. The package should verify before returning that it can access the framework, that it recognizes the context, and that it is compatible with the viewing environment. Any additional initialization may take place asynchronously. When invoked by the Framework, the initialize() method shall change its state to `INIT` and signal the state change to the framework invoking the Framework's `Framework.status()` method with a `StatusDescriptor` containing a code of `CC_STATE_CHANGE`. The Framework will then remain in the `PENDING` state until the Package signals the success or failure of the initialization process by setting its state accordingly and again invoking the Framework's `Framework.status()` method.

A Package will be provided with an instance of a `container` in which it is expected to construct its user interface. The class of the `container` is dependent on the viewing environment. For example, in an environment using HTML and browsers, the container may be an HTML DOM element (e.g., a <div> node). In the case of an Android app being used as the viewing environment, the container may be an instance of the `Fragment` class while for iOS apps the `UIView` class would be used.

Usage:

```
    initialize(context, framework, container)
```

Parameters:

`context`:	the retailer's context
`framework`:	framework instance implementing the `Framework` interface as defined in this document.
`container`:	UI component in which the package will be displayed. The class of the `container` is dependent on the viewing environment.

Returns:

`StatusDescriptor` with one of the following codes:

```
CC_INVALID_CONTEXT
CC_UNSUPPORTED_ENVIRONMENT
CC_OP_FAILED
CC_IN_PROGRESS
CC_COMPLETED
```

ECMAScript Example

4.3.1.2 enable()

Indicates to the package that it is being given control of the user interface and that it is now visible to the consumer.

Usage

```
enable()
```

Parameters: none

Returns: none

ECMAScript Example

4.3.1.3 disable()

Indicates to the package that control of the user interface is reverting to the framework and that it is no longer visible to the consumer. A package that has been disabled may be re-enabled at some point in the future, hence the current state should be preserved.

When disable() is invoked, the Package should dispose of resources, save state, and perform any other function that would leave the Package in a statue where it could later be enabled or terminated.

Usage

```
disable()
```

Parameters: none

Returns: none

ECMAScript Example

4.3.1.4 terminate()

Signals to the Package that the Framework intends to take back control of the user interface and permanently end the activities of the package. The package should immediately perform shutdown and clean-up procedures. All session data that the package wishes to persist should be immediately saved. Any remote resources should be freed up.

A Package should perform any termination-related actions asynchronously. Implementations of the `terminate()` function should return immediately using the return value to indicate if further actions will be taken asynchronously.

The amount of time that a framework allows a package to take before all resources are garbage-collected is unspecified and entirely up to the framework implementation. Once the `terminate()` signal has been given, the framework is no longer obligated to respond to any calls from the package.

Usage

```
terminate()
```

Parameters: none

Returns:

Boolean indicating if the package has completed all required actions or intends to carry out additional clean-up procedures.

ECMAScript Example

4.3.1.5 getState()

Requests the current state of the Package.

Typically this method is invoked by the framework after receiving a `StatusDescriptor` with a code indicating a state change has taken place. A framework may, however, request the package state at any time. For example, the framework may periodically request the package state to verify it is still functioning and has not crashed.

Usage

```
getState()
```

Parameters: none

Returns:

```
PackageState::= {LOADED | INIT | AVAILABLE | RUNNING| FAILED | EXITING
| TERMINATED }
```

ECMAScript Example

4.3.1.6 getExperienceId()

Requests the identifier of the *experience* provided by the Package. This is a string value that the framework may use as a key when storing or retrieving information relating to how a consumer has been interacting with the package (e.g., bookmarks, preferences, last playback state).

A content provider *may* choose to assign a unique experienceId to each package. Alternatively, multiple packages may be created, all of which are intended to provide the same experience (e.g., one built around *Sita Sings the Blues*) but each targeted for different viewing environments or with different layouts and background graphics. In that type of situation, use of a common experienceID allows heterogenous packages to provide a uniform and common experience despite package upgrades changes.

Usage

```
getExperienceId()
```

Parameters: none

Returns:

```
id: String
```

ECMAScript Example

4.3.2 Connectivity Subgroup

Method Summary	
`connectivityChange`	Signals the Package that either the state of a network connection has changed or that the network being used has changed

4.3.2.1 connectivityChange()

Signals to the package that either the state of a network connection has changed or that the network being used has changed.

Usage

```
connectivityChange(ConnectivityState state)
```

Parameters:

`State`: a `ConnectivityState` instance

Returns: none

ECMAScript Example

4.3.3 Environment Subgroup

Method Summary	
`deviceStatusChange`	Signals to the package that there has been a change in the status, state, or capabilities of the device in which the package is running.

4.3.3.1 deviceStatusChange()

Signals to the Package that there has been a change in the status, state, or capabilities of the device in which the package is running. This is a most often an occurrence on mobile devices but the API does not preclude the use of this method by frameworks running in other viewing environments.

Usage

```
deviceStatusChange()
```

Parameters: none

Returns:

key/value pair array

ECMAScript Example

4.4 Shared Data Structures

4.4.1 Connectivity State

This structure may be used to pass information regarding the current state of network connectivity from the framework to the package. Connectivity is defined in terms of four properties:

- The type of network being accessed
- Current state of the connection to the network
- The reliability of the connection
- The subjective through-put of the connection

Both *reliability* and *throughput* are specified as capability levels using

```
NET_CAPABILITY ::= {LOW | MEDIUM | HIGH  | UNKNOWN}.
```

These are treated by the API as relative and subjective valuations and no attempt is made to equate them to absolute values. The other two properties are defined using the following constants:

```
NET_STATE ::= {CONNECTED | CONNECTING | DISCONNECTED | UNKNOWN }
NET_TYPE ::= {BLUETOOTH|WIRE|MOBILE|WIFI|WIMAX|NONE|UNKNOWN}
```

Connectivity state is obtained by the framework from the viewing environment by means of the viewing environment's native API. From the perspective of the package it is, therefore, a read-only construct. If a device has multiple network connections, the information provided will pertain to the connection being used to access or download content or to interface to backend services.

Field Summary	
`type`	the type of network being used to connect the viewing environment to the Internet.

`connectionState`	the current state of the network connection being used to connect the framework and package to the Internet.
`reliability`	an assessment of the reliability of the network connection.
`throughput`	an assessment of the reliability of the network connection's delivered bandwidth for downloads.

4.4.1.1 type

Specifies the type of network being used to connect the framework and package to the Internet. If the device is disconnected a value of NONE will be returned. Any form of non-RF based connection (e.g., an IEEE 802.3 LAN) will be designated as `WIRE`.

```
NET_TYPE ::= {BLUETOOTH|WIRE|MOBILE|WIFI|WIMAX|NONE|UNKNOWN}
```

4.4.1.2 connectionState

Specifies the current state of the network connection being used to connect the framework and package to the Internet.

```
NET_STATE ::= {CONNECTED | CONNECTING | DISCONNECTED | UNKNOWN }
```

4.4.1.3 reliability

Contains an assessment of the reliability of the network connection. This should be regarded as a relative and subjective valuation. It is up to the framework implementer to determine the methodology, metrics, and time-span that is used to perform the assessment. Any framework that does not provide this capability should specify a valuation of `UNKNOWN`.

```
NET_CAPABILITY ::= {LOW | MEDIUM | HIGH  | UNKNOWN}
```

4.4.1.4 throughput

Contains an assessment of the reliability of the network connection's delivered bandwidth for downloads. This should be regarded as a relative and subjective valuation. It is up to the framework implementer to determine the methodology, metrics, and time-span that is used to perform the assessment. Any framework that does not provide this capability should specify a valuation of `UNKNOWN`.

```
NET_CAPABILITY ::= {LOW | MEDIUM | HIGH  | UNKNOWN}
```

4.4.2 Status Descriptor

This structure is used to exchange status and error notifications. A `StatusDescriptor` instance must, at a minimum, specify the following values:

`level` - severity of condition being reported
`completionCode` – indicates the final status of the activity whose status is being reported.

Two optional fields are also provided that may be used to include textual descriptions and details for inclusion in log message or for display to users.

A *context object* may also be included in a status descriptor. This may be used to provide data indicating the specific nature of the operation whose status is being reported (e.g., the URL of a file that was being downloaded).

Completion Code	Description
CC_OP_CANCELLED	Requested operation has been cancelled
CC_OP_COMPLETED	Requested operation has completed. Results now available.
CC_OP_FAILED	Requested operation has failed
CC_OP_PENDING	Requested operation has not yet been initiated and is still pending
CC_OP_IN_PROGRESS	Requested operation has been initiated and is proceeding
CC_STATE_CHANGE	Package state has changed.
CC_INVALID_PARAM	Service provider does not recognize or support a parameter value
CC_UNSUPPORTED	Request or operation not supported in current environment

Severity	Description
LEVEL_INFO	Informative or diagnostic. No impact on capabilities or behavior.
LEVEL_WARNING	Unexpected event. No observable impact on capabilities or behavior.
LEVEL_ERROR	Undesired impact on capabilities or behavior.
LEVEL_FATAL	Unable to continue normal activities in support of user.

Field Summary	
level	the severity level associated with the status being reported.
completionCode	a code indicating the final status of the activity associated with the status being reported.
message	a brief summary message that is appropriate for use in logging or in a status message displayed to a consumer

`details`	a text message that may provide background details that clarify the summary message.
`context`	an Object that in some way provides the receiving entity with additional context as to what the status message is in regards to

4.4.2.1 level

Specifies the severity level associated with the status being reported.

```
LEVEL ::= {INFO | WARNING | ERROR  | FATAL}
```

4.4.2.2 completionCode

Contains a code indicating the final status of the activity associated with the status being reported.

`code:` integer value equal to one of the `CC_xxxx` values

4.4.2.3 message

Contains a brief summary message that is appropriate for use in logging or in a status message displayed to a consumer (e.g. “Network unavailable. Try again later”).

Note that as these messages are user-visible, the returned strings must be localized.

`message`: text String

4.4.2.4 details

Contains a text message that may provide background details that clarify the summary message.

Note that as these messages are user-visible, the returned strings must be localized.

`details`: text String

4.4.2.5 context

Contains an Object that in some way provides the receiving entity with additional context as to what the status message is in regards to. Examples might be a String with a request ID or an UI container instance.

`context`: Object instance

5 CONTENT ACCESS API GROUP

Content Access APIs are used by a package to obtain or determine the accessibility of specific content (i.e., media files and streams). This includes determining which content the user has the rights to access, providing the user a means to acquire additional content (e.g., 'Buy' or 'Rent' button) and, where applicable, the means to download content. Notifications, such as download complete, are also included.

Functionality to be implemented by the Framework is divided into the following subgroups:

Subgroup Summary	
Availability	Provides information about whether the user already has access to content, whether they can obtain access to content, or whether there is a transaction in progress.
Access Event	Methods for a Package to register for, and receive, event notifications.
Download	Methods to control file download

A Package will only implement the Access Event subgroup.

5.1 Content Access Codes

These codes are used when inquiries are made as to the accessibility of a content item or when an event is associated with some change in accessibility (see Sections 5.2.1 and 5.3.1.1 respectively). The entitlement codes are also used when the package attempts to acquire access to content (see Section 5.2.1.2).

Entitlement	Description
`ACC_AVAIL_2BUY`	The content may be bought from this Retailer
`ACC_AVAIL_2RENT`	The content may be rented from this Retailer
`ACC_AVAIL_FREE`	The content does not require purchase or rental in order to access
`ACC_AVAIL_BOUGHT`	The content has been bought from this Retailer
`ACC_AVAIL_RENTED`	The content has been rented from this Retailer
`ACC_AVAIL_PENDING`	A purchase or rental has been initiated and a change in availability is therefore pending.
`ACC_AVAIL_UNAVAIL`	The content is not available to the Consumer
`ACC_AVAIL_UNK`	Availability cannot be determined at this time
`ACC_CONTENT_UNK`	Unrecognized `contentId`

5.2 Framework Interface

Framework developers are required to provide an implementation of the `ContentMgr` interface. This defines the methods that a package may use to determine the availability of, and gain access to, specific content files.

Method Summary	
`getAvailability`	Checks the consumer's access rights to the specified content from the Retailer.
`acquire`	Opens the retailers purchase UI allowing the user to buy or rent the desired item.
`addListener`	Adds the listener to the listener list.
`removeListener`	Removes the listener from the listener list.
`download`	Requests that a download be initiated of the specified content.
`cancelDownload`	Cancel a previously requested and still incomplete download.

5.2.1 Availability Subgroup

Content availability APIs provide information about whether the user already has access to content, whether they can obtain access to content, or whether there is a transaction in progress. If the user does not have access, an API is provided to initiate a transaction to obtain content.

5.2.1.1 getAvailability()

Checks the consumer's access rights to the specified content from the Retailer. This may include identification of any restrictions that may apply. Possible restrictions might include if it must be purchased or rented prior to playback, if there is a time limit on the availability, or if a limited number of playbacks are allowed.

Content may be available in more than one variant (e.g., director's cut and theatrical). A query may, therefore, return the availability data for multiple content, each with its own unique `contentId.`

The query completes asynchronously. Results will be returned asynchronously via a callback function. The arguments passed when invoking the callback are an associative array of one or more availability records, where the *key* is a `contentId` and the *value* is an array containing all applicable codes. For example:

- The consumer has purchased movie "A".
- The consumer rented movie "B", and it is also available for purchase
- Movie "C" is available for rental but not purchase.

Usage

```
getAvailability(String contentId, function callback)
```

Parameters:

`contentId`: - the retailer's content identifier

`callback`: - function to be passed the final results.

Returns:

`StatusDescriptor` : with `completionCode`

ECMAScript Example

```
getAvailability(_contentId, function(  _contentId, _carData) {

    var _carArray = _carData[_contentId];

    var available = ($.inArray(ACC_AVAIL_RENTED, _carArray) >= 0
        || $.inArray(ACC_AVAIL_BOUGHT, _carArray) >= 0 ||
            $.inArray(ACC_AVAIL_FREE, _carArray) >= 0);

    });
```

5.2.1.2 acquire()

Opens the retailers purchase UI allowing the user to buy or rent the desired item. Purchasing is asynchronous, hence this method will return immediately. Upon completion any registered listeners will be informed by the Framework of the change in content availability status.

Usage

```
acquire(String contentId, int request, String requestId)
```

Parameters:

`contentId`: - the retailer's content identifier

`request`: - identifies whether the intent is to buy or rent. A value of `ACC_AVAIL_2BUY` or `ACC_AVAIL_2RENT` must be specified.

`requestId`: - identifier to be provided with any event notification associated with this request

Returns:

`StatusDescriptor` instance with the `requestId` as the `context` value.

ECMAScript Example

5.2.2 Access Event Subgroup

5.2.2.1 addListener()

Adds the listener to the listener list. The listener is registered for all event notifications associated with this interface.

Usage

```
addListener(AccessEventListener listener)
```

Parameters:

`listener`

Returns:

`true` if the listener was added successfully

ECMAScript Example

5.2.2.2 removeListener()

Removes the listener from the listener list. This removes an `AccessEventListener` that was previously registered for all event notifications.

Usage

```
removeListener(AccessEventListener listener)
```

Parameters:

`listener`

Returns: `none`

ECMAScript Example

`true` if the listener was successfully removed

5.2.3 Download Subgroup

The download(), cancelDownload() and canDownload() methods are provided to control file download. The download function, if available, is provided in the Framework.

The Package must first invoke canDownload() to determine if the Framework is capable of downloading content. If the Framework is capable and the Package wishes to start a download, download() is invoked, including the `contentID` of the content to be downloaded.

To cancel a download, cancelDownload() is used. It is assumed the Framework will make a best effort to cancel the download.

We are currently considering adding the means to monitor the status of a download, including an indication of download completion. This is TBD.

5.2.3.1 download()

Requests that a download be initiated of the specified content. Each retailer may have differing procedures and options for downloading. For example, Retailer 'A' may allow the consumer to select the location (i.e., directory) to which the content will be transferred while Retailer 'B' may not offer that degree of flexibility. Any consumer inputs, dialogs, or interactions subsequent to initiation of the download request are, therefore, the responsibility of the framework's `ContentMgr` implementation.

Usage

```
download(String contentId, String requested)
```

Parameters:

`contentId`: - the retailer's content identifier

`requestId`: - identifier to be provided with any event notification associated with this request (optional)

Returns:

`StatusDescriptor` instance with the `requestId` as the `context` value

ECMAScript Example

5.2.3.2 cancelDownload()

Cancel a previously requested and still incomplete download. The operation to cancel is indicated by the `requestId` which is expected to match that used in the original download request. Recovery of any allocated resources (i.e., storage space) is the responsibility of the framework.

The status code that will be returned to the caller should be interpreted as follows:

- A successful cancellation will be indicated by a code of CODE_OP_CANCELLED
- In the event that the download operation has already completed, a code of CODE_OP_COMPLETED is returned. This indicates that the content currently resides in the targeted download location but can be deleted via a removeContent() request.
- If the requestId is unknown, a code of CODE_OP_INVALID_PARAMETER is returned.

Cancellation of a download does **not** eliminate the need to notify any listeners of the conclusion of a download operation. The status code of `CODE_OP_CANCELLED` that will be returned to the caller *is in addition to* the status code of `CODE_OP_CANCELLED` that will be provided in any event notifications. This means that if the entity invoking the `cancelDownload` method is also a listener, it should expect to receive redundant information via an eventual event notification of the download's cancellation.

Usage

`cancelDownload(String requestId)`

Parameters:

`requestId`: - identifier to be provided with any event notification associated with this request

Returns:

`StatusDescriptor` instance with the `requestId` as the `context` value

ECMAScript Example

5.2.3.3 canDownload()

Returns an indication of the Framework's ability to download media. An inability to download is indicated by a return value of `false` and may be due to one or more of the following conditions"

- capability is not provided by the Framework or has been disabled by the user
- the device is not connected to a network
- required file storage space is not available on the device

Usage

`canDownload()`

Parameters: `none`

Returns:

`true` if the framework is capable of downloading media to the user's device.

ECMAScript Example

5.3 Package Interface

5.3.1 Access Event Subgroup

Components wishing to receive notification of account-related events must register as `AccessEventListeners`. While it is assumed that a Package will implement the `AccessEventListener` interface, this is not a requirement of the API

5.3.1.1 eventNotification()

Informs a registered listener of a change in content availability status, the completion of a user initiated request, or both. Changes to content availability may be due to a request from the consumer (e.g., a purchase) or may result from some event or action not initiated by the consumer (e.g., expiration of a rental period). If the change is derived from a consumer initiated request, any requestId provided when the acquire() method was invoked will be included in the event notification.

Any event notification associated with a request made via the acquire() method will also indicate the completion status of the request. In these cases the requestStatus field will be an integer value equal to one of the StatusDescriptor CODE_OP_xxxx values (see Section 4.4.1).

Usage

```
eventNotification(String contentId, int[] eventCode, String
        requestId, int requestStatus)
```

Parameters:

contentId: the retailer's content identifier

eventCode: the set of all applicable availability codes.

requestId: identifier associated with the request that resulted in the change to availability status (optional)

requestStatus: integer value equal to one of the CODE_OP_xxxx values (optional)

Returns: none

ECMAScript Example

6 ACCOUNT ACCESS API GROUP

Account Access allows a package to access account properties, such as screen name, preferences and favorites as appropriate. It also provides the means to determine if the user is signed in. If appropriate, it can be used by the package to indicate that the user needs to sign in or sign out. The sign-in/sign-out procedures, like all other aspects of account management, are a retailer-specific process and are therefore handled by the framework.

Functionality to be implemented by the Framework is divided into the following subgroups:

Subgroup Summary	
`Basic`	Methods to sign in or out and to access consumer preferences
`Account Event`	Interface to receive notification of account-related events

A Package will only implement the Account Event subgroup.

6.1 Framework Interface

6.1.1 Basic Subgroup

The role of the Framework instance is primarily to provide the package with access to user account data.

Method Summary	
`signIn`	Opens the retailer's sign-in UI.
`signOut`	Initiates the retailer's sign-out process.
`isSignedIn`	Indicates if consumer is currently signed in.
`getAccountProperties`	Request for information regarding the properties of a signed-in consumer.
`getPreferences`	Request for information regarding the user interface preferences of a signed-in consumer.

6.1.1.1 signIn()

Opens the retailer's sign-in UI. The specifics by which the consumer establishes their identity is up to the retailer. The sign-in process should be handled asynchronously. Thus upon the completion of this method the returned `StatusDescriptor` will indicate only if the framework was able to initiate the sign-in process. An asynchronous event notification will be sent to all registered listeners when the sign-in process completes. One implication of this behavior is that the calling package will *not* receive an event notification if the user fails to complete the sign-in process.

Usage

`signIn(String requestId)`

Parameters:

`requestId`: Identifier that will be associated with any event resulting from the request

Returns:

`StatusDescriptor` instance

ECMAScript Example

6.1.1.2 signOut()

Initiates the retailer’s sign-out process. An asynchronous event notification (`ACCNT_SIGNED_OUT`.) is associated with the sign-out process.. Since sign-out may potentially not have involved invocation of the `signOut()` method, the event notification may not always have an associated `requestId`.

Usage

`signOut(String requestId)`

Parameters

`requestId`: Identifier that will be associated with any event resulting from the request

Returns:

`StatusDescriptor` instance

6.1.1.3 isSignedIn()

Query to determine if consumer is currently signed in.

Usage

`isSignedIn(function callback)`

Parameters:

`callback`: - function to be passed the final results. This will be a Boolean set to `true` if consumer is currently signed in

Returns:

`StatusDescriptor` instance indicating `CC_OP_PENDING`

ECMAScript Example

6.1.1.4 getAccountProperties ()

Request for information regarding the properties of a signed-in consumer (i.e., screen name, age, avatar, etc.). Properties are specified as a JSON-compatible object. The available properties are specific to each retailer. Account properties are, for a package, immutable and read-only. That is to say, only the retailer's framework may include or modify the account properties.

Usage

```
getAccountProperties()
```

Parameters: none

Returns:

`properties` –a JSON-compatible object

6.1.1.5 getPreferences()

Request for information regarding the user interface preferences of a signed-in consumer (i.e., layout format, language, font size). Preferences are specified as a JSON-compatible object. The preferences specified may be specific to each retailer's framework. Unlike the `AccountProperties,` the preferences are mutable.

Usage

```
getPreferences()
```

Parameters: none

Returns:

`preferences` –a JSON-compatible object

ECMAScript Example

6.1.2 Account Event Subgroup

6.1.2.1 addListener()

Adds the listener to the listener list. The listener is registered for all event notifications associated with this interface.

Usage

```
addListener(AccountEventListener listener)
```

Parameters:

`listener`

Returns:

`true` if the listener was added successfully

ECMAScript Example

6.1.2.2 removeListener()

Removes the listener from the listener list. This removes an AccountEventListener that was previously registered for all event notifications.

Usage

```
removeListener(AccountEventListener listener)
```

Parameters:

```
listener
```

Returns: `none`

ECMAScript Example

`true` if the listener was successfully removed

6.2 Package Interface

Components wishing to receive notification of account-related events must register as `AccountEventListeners`. While it is assumed that a Package will implement the `AccountEventListeners` interface, this is not a requirement of the API.

6.2.1 Account Event Subgroup

6.2.1.1 Account Event Codes

The following event types may be communicated to an `AccountEventListeners`:

Event	Description
`ACCNT_SIGNED_IN`	Indicates the user has signed in.
`ACCNT_SIGNED_OUT`	Indicates the user has been signed out.
`ACCNT_PREF_CHANGE`	Indicates a change to the set of user preferences.

6.2.1.2 eventNotification()

Notification of an event related to the currently active account.

The only event-type currently supported is `ACCNT_SIGNED_OUT`. This event is associated with the consumer being ‘signed out’ either by user request or by software initiative (e.g., session timed-out due to inactivity).

Usage

```
eventNotification(int eventCode, String requestId)
```

Parameters:

`eventCode`**:** integer value equal to one of the `ACCNT_xxxx` values.

`requestId`: (optional) indicates if the event was generated due to a `signIn` or `signOut` request and, if so, which request.

Returns: none

ECMAScript Example

7 PLAYER INTERACTION API GROUP

The Player Interaction APIs provide the means to play content and to interact with the user while playing content. The framework is responsible for providing a media player capability. The package may use the API to manage the life-cycle of one or more players.

7.1 Overview

7.1.1 Component Model

A Package responds to a consumer's request to play a media stream by obtaining access to a Player. This access is obtained by means of the Framework's Lifecycle subgroup of functions. These allow the Package to ask the Framework to create, or dispose of, Players.

An instantiated `Player` will contain both a `Video` and an `Audio` subcomponent. A `MediaDescriptor` element provides information regarding the content currently associated with the `Player`, as well as the current state of playback. The Package may control the Player using the functions supported by the Basic and Trickplay subgroups.

Event handling and callback notification is provided via registered listeners. A package should, therefore, include one or more components that implement the `PlayerEventListener` interface. This must be explicitly registered with the framework using the `addListener()` method.

Subgroup Summary	
Lifecycle	Methods to create and destroy a Player instance
Basic	Methods that allow minimal required direct control of media playback
Trickplay	Trickplay functions such as fast forward and reverse
Controls	Enable or Disable accessibility of Player's Control UI
Sound	Methods to control audio output
TrackSelection	*[NOTE: We are currently gathering requirements for this subgroup. This subgroup will provide the means to select track on playback based on track selection algorithms within the Package.]*
Player Event	Interface to receive notification of player-related events
Geometry	Methods to control size and/or orientation of video display

All sub-groups are to be supported by all players and frameworks. A package will support only the Player Event subgroup.

7.1.2 Control of Media Stream

A package may set the direction of play (i.e., forward or reverse) as well as the speed of play. Playback is then controlled via a start/pause/stop sequence of interactions. The API also allows a package to initiate a transition from the current location to new location (e.g., 'jump back 15 seconds').

A package may expect these basic operations to be supported by all players and frameworks. The ability of a framework's player to execute a requested action may, however, be limited for several reasons:

- The action is supported but not in the player's current state (e.g., a request to STOP a player that is already stopped).
- The action is supported but one or more of the parameters are not supported (e.g., a request to play in reverse at a speed not supported in that direction).

For this reason, methods invoked by the package will have a return code set by the framework that indicates the success or failure of the framework in imitating the requested action. If the framework was unable to initiate, the return code will indicate the reason (e.g. `RESP_INVALID_STATE`).

7.2 Framework Interface

The framework side of the player interface defines the life-cycle methods as well methods that allow direct control of media playback via the software. This mode of control is an alternative to direct control of playback by the Consumer via a player's internal control panel, assuming it provides one. The API allows the player's controls to be hidden or disabled if the software wishes to impose restrictions on the consumer's actions (e.g., temporarily disabling fast-forward).

7.2.1 Shared Constants

7.2.1.1 Response Codes

The following may be used as return codes by methods provided by the Player interface.

Code	Description
`RESP_OK`	Indicates the player has begun streaming the media. The direction and rate of playback is not specified.
`RESP_INVALID_STATE`	Indicates the requested operation may not be performed in the Player's current state.
`RESP_UNSUPPORTED_CMD`	Indicates the requested operation is not supported by the Player.
`RESP_UNSUPPORTED_OPT`	Indicates the requested operation is supported by the Player but a parameter value was not.

`RESP_FAILED`	Indicates the requested operation was attempted but not successful.

7.2.2 Lifecycle Subgroup

Method Summary	
`createPlayer`	Create a Player instance that is able to play the desired media item.
`destroyPlayer`	Indicates to the framework that the package has no further use for the player

7.2.2.1 createPlayer()

Create a Player instance that is able to play the desired media item. The Framework will return a Player instance if it is able to (a) identify the specified content, (b) verify that the Consumer is entitled to view the content, and (c) provide a player that is compatible with the content's format and characteristics. It will then be the responsibility of the Package to configure and initialize the player using the Controls, Sound, and Geometry subgroups.

The return value will be a `StatusDescriptor.` If the framework is able to provide a player it will be returned to the caller as the `context` Object of the `StatusDescriptor`.

Usage

```
createPlayer(String contentId, String playerId, Boolean advanced)
```

Parameters:

`contentId`: The retailer's content identifier

`playerId`: ID assigned by package to player instance

`advanced`: if `true` a player with support for the Trickplay subgroup should be returned. If the framework does not support this capability, or if the argument is `false`, a basic player is returned.

Returns:

`StatusDescriptor`

ECMAScript Example

7.2.2.2 destroyPlayer()

Indicates to the framework that the package has no further use for the player and that clean-up and garbage collection may proceed.

Usage

```
destroyPlayer(Player playerId )
```

Parameters:

playerId : ID of the Player instance to be disposed of. Should match ID used when calling `createPlayer()`

Returns: none

ECMAScript Example

7.2.3 Basic Subgroup

This subgroup provides the minimal and essential functions needed to control playback.

Subgroup	Method Summary	
Basic	`play`	Start playback using currently specified rate and direction.
	`togglePause`	Restart or pause playback depending on current state.
	`setPaused`	Restart or pause playback depending on specified flag.
	`isPaused`	Returns a Boolean flag indicating if the player is currently paused.
	`stop`	Stops playback and resets to the media stream to its initial state.
	`setPoster`	Set image shown prior to start of play
	`isVisible`	Query to determine if player is currently visible to user
	`jumpTo`	Change location in the media stream to the specified location. This is an immediate shift and the user will not see intermediate frames displayed.
	`getCurrentTime`	Returns the current playback location in seconds
	`supportsApiOption`	Returns Boolean indicating if player instance supports an optional capability.

7.2.3.1 Events

This section has been removed. Refer to Section 7.3.1.1

7.2.3.2 play()

Start playback using currently specified rate and direction.

Usage

```
play()
```

Parameters: none

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.3.3 togglePause()

Restart or pause playback depending on current state. If restarting, playback will resume at last location using currently specified rate and direction.

Usage

```
togglePause()
```

Parameters: none

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.3.4 setPaused()

Restart or pause playback depending on specified flag. If restarting, playback will resume at last location using the currently specified rate and direction.

Usage

```
setPaused(Boolean pause)
```

Parameters:

`pause` - `true` if this component should be paused, false otherwise

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.3.5 isPaused()

Returns a Boolean flag indicating if the player is currently paused.

Usage

`isPaused()`

Parameters: none

Returns:

`true` if the player is currently paused, false otherwise

ECMAScript Example

7.2.3.6 stop()

Stops playback and resets to the media stream to its initial state.

Usage

`stop()`

Parameters: none

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.3.7 setPoster()

Set the image that will be displayed while media is loading and prior to the start of play. Setting poster after play has begun will have indeterminate consequences in that the change may or may not be visible at some future point.

Usage

`setPoster(String imageUrl )`

Parameters:

`imageUrl`: path to image

Returns: none

ECMAScript Example

7.2.3.8 isVisible()

Query to get the visibility status of the player. A player may be hidden from the user even when the browser window is not (e.g., when page tab is not selected). Typically this method will invoked by the package in response to having received a PE_VISIBILITY event from the framework.

Usage

```
isVisible()
```

Parameters: none

Returns:

`true` if the player is currently visible to user, false otherwise

ECMAScript Example

7.2.3.9 jumpTo()

Change location in the media stream to the specified location. This is an immediate shift and the user will not see intermediate frames displayed. The state of the player will be restored upon completion of the operation (i.e., if the player was paused prior to the jump command it will still be paused afterwards but if it was playing backwards at half normal rate it will still do so afterwards).

Usage

```
jumpTo(int position)
```

Parameters:

`int position` absolute position in time given in milliseconds

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.3.10 getCurrentTime()

Get the current position in the playback stream. The returned value will be rounded off to the nearest second.).

Usage

```
getCurrentTime()
```

Parameters: none

Returns:

`int position` absolute position in time given in seconds.

ECMAScript Example

7.2.3.11 supportsApiOption()

Query to determine if a given set of optional functions is supported by the player instance. Functionality is identified by a string argument specifying the name of one of the CPEx Player API subgroups. Note that names are case sensitive.

Usage

```
supportsApiOption(String apiGroup)
```

Parameters:

apiGroup: String identifying an optional subgroup `[Trickplay | Controls | Sound]`

Returns:

`true` if the player supports the API sub-group, false otherwise

ECMAScript Example

7.2.4 Trickplay Subgroup

This subgroup provides access to more advanced control of the playback stream.

Subgroup	Method Summary	
Trickplay	`increaseRate`	Increase rate to next highest available.
	`decreaseRate`	Reduce rate to next slower rate available.
	`setRate`	Set the media playback rate to the value specified.
	`getRate`	Returns the current rate of media playback.
	`setReversed`	Set the direction of playback.
	`isReversed`	Returns a Boolean flag indicating if the direction of playback is currently reversed.
	`setPlaybackMode`	Set both the direction and speed of playback.
	`jump`	Change location in the media stream by the specified amount. This is an immediate shift and the user will not see intermediate frames displayed.

7.2.4.1 Events

The following event types may be communicated to a `PlayerEventListener`:

Event	Description

`PE_MODE_CHANGE`	Indicates a change to the rate or direction of playback.
`PE_REPOSITION_START`	Indicates that the player has an initiated a repositioning of the point of playback in response to a jump command.
`PE_REPOSITION_END`	Indicates that the player has an completed or terminated a repositioning of the point of playback in response to a jump command.

7.2.4.2 Trickplay Constants

7.2.4.2.1 Playback Rates

The following may be used to indicate the rate of media playback. Playback rate is specified independently of the direction of playback (i.e., *forward* or *reverse*). A specific player may or may not support a given rate or may only support that rate when used in one direction (e.g., a player allows for half-speed playback in the forward direction only).

Code	Description
`RATE_QUARTER`	¼ of normal speed
`RATE_HALF`	½ of normal speed
`RATE_NORMAL`	normal speed (default); the FPS equivalent is media-dependant.
`RATE_DOUBLE`	2x the normal speed
`RATE_TRIPLE`	3x the normal speed
`RATE_SLOWEST`	Slowest rate supported by the player
`RATE_FASTEST`	Fastest rate supported by the player

7.2.4.3 increaseRate()

Increase rate to next highest available. The return code will indicate the new rate. If the rate is already at the highest available setting it will remain so and the return code will be `RATE_FASTEST`. The availability of a specific rate may be dependant on the capabilities of a player as well as the current direction of playback.

Usage

```
    increaseRate()
```

Parameters: none

Returns:

`rate:` integer value equal to one of the `RATE_xxxx` values.

ECMAScript Example

7.2.4.4 decreaseRate()

Reduce rate to next slower rate available. The return code will indicate the new rate. If the rate is already at the slowest available setting it will remain so and the return code will be `RATE_SLOWEST`. The availability of a specific rate may be dependant on the capabilities of a player as well as the current direction of playback.

Usage

```
decreaseRate()
```

Parameters: none

Returns:

`rate:` integer value equal to one of the `RATE_xxxx` values.

ECMAScript Example

7.2.4.5 setRate()

Set the media playback rate to the value specified. The availability of a specific rate may be dependent on the capabilities of a player as well as the current direction of playback.

Usage

```
setRate()
```

Parameters:

`rate:` integer value equal to one of the `RATE_xxxx` values

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.4.6 getRate()

Returns the current rate of media playback.

Usage

```
getRate()
```

Parameters: none

Returns:

`rate:` integer value equal to one of the `RATE_xxxx` values.

ECMAScript Example

7.2.4.7 setReversed()

Set the direction of playback. If `flag==TRUE` then playback is reversed.

Usage

setReversed(Boolean reverse)

Parameters:

`reverse` - `true` if the direction of playback should be reversed, false otherwise

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.4.8 isReversed ()

Returns a Boolean flag indicating if the direction of playback is currently reversed. The direction of playback is treated as a state variable that is independent of whether or not the player is currently playing a media stream. For example, a player that is currently paused will still have a reversed state of `true` or `false`. In these situations the return value indicates the direction of playback that will be used when the player resumes operations.

Usage

`isReversed()`

Parameters: none

Returns:

`true` if the player is currently set to playback media in reverse, false otherwise.

ECMAScript Example

7.2.4.9 setPlaybackMode()

Set both the direction and speed of playback. The availability of a specific rate may be dependant on the capabilities of a player as well as the direction of playback. The player will validate a requested rate based on *requested* direction of playback, rather than the *current* direction of playback. If the combination is not supported the return code will be `RESP_UNSUPPORTED_OPT` and no change will take place.

Usage

`setPlaybackMode(Boolean reversed, int rate)`

Parameters:

`reversed`: if `true` indicates playback direction is reversed.

`rate`: integer value equal to one of the `RATE_xxxx` values.

Returns:

`status`: integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.4.10 jump()

Change location in the media stream by the specified amount. This is an immediate shift and the user will not see intermediate frames displayed. The state of the player will be restored upon completion of the operation (i.e., if the player was paused prior to the jump command it will still be paused afterwards but if it was playing backwards at half normal rate it will still do so afterwards).

Usage

`jump(int offset)`

Parameters:

`int offset` from current position specified as time given in milliseconds.

Returns:

`status`: integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.5 Controls Subgroup

This subgroup allows a package to enable or disable the consumer's access to any control UI that is intrinsic to a player.

Subgroup	Method Summary	
Controls	`setVisibleControls`	Set the visibility of any controls internal to the Player.
	`hasVisibleControls`	Returns a Boolean flag indicating if Player has internal controls currently visible to the user.

7.2.5.1 Events

Event	Description
`PE_CTRLBAR_HIDDEN`	Indicates the player's internal control bar has been hidden
`PE_CTRLBAR_VISIBLE`	Indicates the player's internal control bar has been made visible to the user

7.2.5.2 setVisibleControls()

Set the visibility of any controls internal to the Player. If `show==TRUE` then controls are to be displayed. This operation may fail with a return code of `RESP_UNSUPPORTED_CMD` if the player does not provide internal controls or if the visibility of the controls has been locked.

Usage

```
setVisibleControls(Boolean show)
```

Parameters:

`show` - `true` if controls are to be displayed, false otherwise

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values.

ECMAScript Example

7.2.5.3 hasVisibleControls()

Returns a Boolean flag indicating if Player has internal controls currently visible to the user. If a player lacks such controls, the return value shall be `FALSE`.

Usage

```
hasVisibleControls()
```

Parameters: none

Returns:

`true` if the player is currently displaying internal controls, false otherwise

ECMAScript Example

7.2.6 Sound Subgroup

This subgroup allows the Package to read and adjust Player volume and muting.

Subgroup	Method Summary

Sound	`setVolume`	Set the audio volume, from 0.0 (silent) to 1.0 (loudest).
	`getVolume`	Returns the current volume.
	`setMuted`	Mute or unmute the audio.
	`isMuted`	Returns a Boolean flag indicating if the audio is currently muted.

7.2.6.1 Events

The following event types may be communicated to a `PlayerEventListener`:

Event	Description
`PE_MUTE_CHANGE`	Indicates a switch into, or out of, the MUTED AUDIO state.

7.2.6.2 setVolume(float volume)

Set the audio volume.

Usage

```
setVolume()
```

Parameters:

`volume`: floating point value from 0.0 (silent) to 1.0 (loudest).

Returns:

`volume:` value equal to audio volume at conclusion of function call.

ECMAScript Example

7.2.6.3 getVolume()

Get the audio volume setting as a floating point value from 0.0 (silent) to 1.0 (loudest). Note that the value returned is of the volume setting and that muting is considered an independent property. This means that a non-zero volume setting does not mean the user will actually be able to hear any audio output as the audio may be currently muted.

Usage

```
getVolume()
```

Parameters: none

Returns:

`volume`: floating point value from 0.0 (silent) to 1.0 (loudest).

ECMAScript Example

7.2.6.4 isMuted()

Returns a Boolean flag indicating if the audio output is currently muted.

Usage

```
isMuted()
```

Parameters: `none`

Returns:

`true` if the audio output is currently muted, false otherwise.

ECMAScript Example

7.2.6.5 setMuted()

Mute or unmute audio output. Muting is considered to be independent of the *volume* property. This means that specifying `muted` as FALSE does not mean the user will actually be able to hear any audio output as the audio volume currently may be set to 0.0.

Usage

```
setMuted(Boolean muted)
```

Parameters:

`Mute – true` if the audio is to be muted, false otherwise.

Returns:

`status:` integer value equal to one of the `RESP_xxxx` value

ECMAScript Example

7.2.7 Track Selection Subgroup

This section is TBD.

7.2.8 Player Event Subgroup

Subgroup	Method Summary	
	`addListener`	Add a PlayerEventListener to the listener list.

Player Event	`removeListener`	Remove the listener from the listener list.
	`requestNotificationAt`	Requests notification event when a specified location in the playback stream is reached.
	`cancelNotificationAt`	Cancels previously requested notification event for a specified location in the playback stream.
	`requestPeriodic`	Requests notification whenever there is a change in the current playback position by some fixed period of time.
	`cancelPeriodic`	Cancels notification at fixed intervals
	`eventNotification`	Notification of an event related to a player.

7.2.8.1 addListener()

Add the listener to the listener list. The listener is registered for all event notifications.

Usage

```
addListener(PlayerEventListener listener)
```

Parameters:

```
listener
```

Returns:

`true` if the listener was added successfully, false otherwise

ECMAScript Example

7.2.8.2 removeListener()

Remove the listener from the listener list. This removes a `PlayerEventListener` that was registered for all event notifications.

Usage

```
removeListener(PlayerEventListener listener)
```

Parameters:

```
listener
```

Returns:

`true` if the listener was removed successfully, false otherwise

ECMAScript Example

7.2.8.3 requestNotificationAt()

Request that any Listeners be notified when playback of the media stream reaches the specified location. The request will remain until cancelled. As a result, if the media is rewound and played thru the specified position again, an additional notification should be delivered.

Usage

```
requestNotificationAt(int position)
```

Parameters:

`int position` absolute position in time given in seconds

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values

ECMAScript Example

7.2.8.4 cancelNotificationAt()

Cancel request to be notified when playback of the media stream reaches the specified location. If there is no currently pending registration for that location, the request will be ignored.

Usage

```
cancelNotificationAt(int position)
```

Parameters:

`int position` absolute position in time given in seconds

Returns:

`status:` integer value equal to one of the `RESP_xxxx` values

ECMAScript Example

7.2.8.5 requestPeriodic()

Request that any Listeners be notified when playback of the media stream has changed by a fixed interval. Notification takes the form of a `PE_TIME_EVENT_P` event. The reporting interval will be up to the framework and will be returned to the caller. The request will remain until cancelled.

Usage

```
requestPeriodic()
```

Parameters: `none`

Returns:

`period`: integer value of reporting interval in milliseconds.

ECMAScript Example

7.2.8.6 cancelPeriodic()

Cancel request to be notified at fixed intervals. If periodic notification is currently off, the request will be ignored.

Usage

```
cancelPeriodic()
```

Parameters: `none`

Returns: `none`

ECMAScript Example

7.2.9 Geometry Subgroup

Subgroup	Method Summary	
Geometry	`setFullscreen`	Set or unset fullscreen mode.
	`isFullscreen`	Returns a Boolean flag indicating if the player is currently in fullscreen mode.
	`setPlayerDimensions`	Set the dimensions of the player
	`getPlayerGeometry`	Get the dimensions of the player and its embedded structures

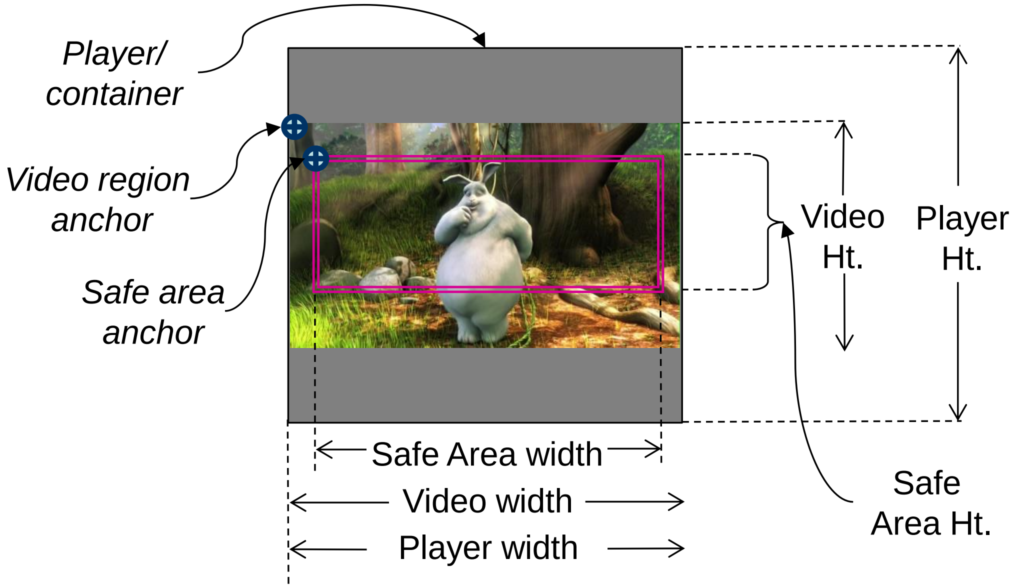
7.2.9.1 Event s

The following event types may be communicated to a `PlayerEventListener`:

Event	Description
`PE_GEOMETRY_CHANGE`	Indicates a change to the player geometry.

7.2.9.2 Player Geometry

The Player Geometry data structure provides a mechanism for a Player instance to communicate all data regarding the geometry of the video presentation area. This includes the size and positioning of designated areas within the player.



The size properties associated with a player differentiate between the size of the player itself and the size of the video being displayed within it. The API allows the setting of the player height and width only. The actual height and width of the region displaying the video will be determined based on the player's dimensions, the space used by any visible controls, and the aspect ratio of the video being displayed. The positioning of the video within the player is indicated by an anchor point. This is specified as a pixel offset from the upper-left corner of the player to the upper-left corner of the video region. Most players will normally attempt to center the video as well as maximize its size while still maintaining the required aspect ratio. Thus, under most circumstances the video anchor point will align with the left side of the player. This is not, however, a requirement.

A player may also provide a "safe area" in which overlays and text may be placed on the screen without interfering with the player's internal controls. A safe area may, therefore, overlap the video region. A safe area will be defined in terms of a height, width, and anchor point. The anchor point will be the offset of the upper-left corner of the safe area from the upper-left corner of the video region. If a safe area is not available or supported, the height and width values returned will be zero.

All sizes and dimensions are specified in pixels and all positions are relative to the parent container's anchor point. The translation of the player-centric coordinates to some other frame of reference (e.g., screen, container) is outside the scope of this API. Package developers should, therefore, verify as part of the integration process any assumptions made in this regard (e.g., that the Player will fill it's parent container).

7.2.9.2.1 playerHeight

The height of the player in pixels (`integer`)

7.2.9.2.2 playerWidth

The width of the player in pixels (`integer`)

7.2.9.2.3 videoHeight

The height of the video region in pixels (`integer`)

7.2.9.2.4 videoWidth

The width of the video region in pixels (`integer`)

7.2.9.2.5 videoOffsetX

The horizontal offset of the video region anchor point in pixels. This is measured from the upper-left corner of the video region to the upper-left corner of the player. (`integer`)

7.2.9.2.6 videoOffsetY

The vertical offset of the video region anchor point in pixels. This is measured from the upper-left corner of the video region to the upper-left corner of the player. (`integer`)

7.2.9.2.7 safeAreaHeight

The height of the safe area in pixels (`integer`)

7.2.9.2.8 safeAreaWidth

The width of the safe area in pixels. (`integer`)

7.2.9.2.9 safeAreaOffsetX

The horizontal offset of the safe area anchor point in pixels. This is measured from the upper-left corner of the safe-area to the upper-left corner of the video region. (`integer`)

7.2.9.2.10 safeAreaOffsetY

The vertical offset of the safe area anchor point in pixels. This is measured from the upper-left corner of the safe area to the upper-left corner of the video region. (`integer`)

7.2.9.3 setFullScreen()

Enable or disable the full-screen mode of display. The return value is an instance of the `PlayerGeometry` data structure with resulting player dimensions.

Usage

```
setFullScreen(Boolean enable)
```

Parameters:

`enable` - `true` if player geometry is to expand to full-screen, `false` otherwise

Returns:

`PlayerGeometry` settings

ECMAScript Example

7.2.9.4 isFullScreen()

Returns a Boolean flag indicating if Player is currently displayed in full-screen mode.

Usage

`isFullScreen()`

Parameters: none

Returns:

`true` if the player is currently in full-screen mode, `false` otherwise

ECMAScript Example

7.2.9.5 [Section removed]

7.2.9.6 getPlayerGeometry()

Provide information about the current display settings in the form of an instance of the `PlayerGeometry` data structure. See Section 7.2.9.2 for further details.

Usage

`getPlayerGeometry()`

Parameters: none

Returns:

`PlayerGeometry` current settings

ECMAScript Example

7.3 Package Interface

The Package Interface for Player Interactions contains a single subgroup: Player Event.

7.3.1 Player Event Subgroup

Components wishing to receive notification of player-related events must register as `PlayerEventListeners` with each player that they wish to receive notifications from. While it is assumed that a Package will implement the `PlayerEventListener` interface, this is not a requirement of the API.

7.3.1.1 Player Event Codes

Each of the API subgroups implemented by a Framework will define it's own set of event codes. The following table identifies all possible event types that may be communicated to a `PlayerEventListener`, including those generated by optional subgroups:

Event	Description
`PE_READY`	Indicates media is available to begin play-back
`PE_PLAYING`	Indicates the player has begun streaming the media. The direction and rate of playback is not specified.
`PE_PAUSED`	Indicates the streaming of the media has been paused.
`PE_STOPPED`	Indicates the streaming of the media has been paused, either due to a control input, a fatal error, or because the end of the media file was reached.
`PE_SUSPENDED`	All player activities have been temporarily suspended due to an event or action external to the Framework or Package.
`PE_RESUMED`	Player has been placed back in an active state after being suspended.
`PE_ERROR`	Indicates an error in playback
`PE_MODE_CHANGE`	Indicates a change to the rate or direction of playback.
`PE_GEOMETRY_CHANGE`	Indicates a change to the player geometry.
`PE_CTRLBAR_CHANGE`	Indicates a change to the visibility of the player's internal control bar.
`PE_MUTE_CHANGE`	Indicates a switch into, or out of, the MUTED AUDIO state.
`PE_REPOSITION_START`	Indicates that the player has an initiated a repositioning of the point of playback in response to a jump command.
`PE_REPOSITION_END`	Indicates that the player has an completed or terminated a repositioning of the point of playback in response to a jump command.
`PE_TIME_EVT_P`	Indicates a change in the current playback position by some fixed period of time (e.g. 250 sec.)
`PE_TIME_EVT_R`	Indicates the playback position has reached a specific point (e.g., 15 m 30 s)
`PE_VISIBILITY`	Indicates change in visibility of player to user (e.g., window has been minimized)

7.3.1.2 eventNotification()

Notification of an event related to a player with which the receiving entity has registered as a `PlayerEventListener`.

Usage

```
eventNotification(String playerID, int eventCode, int evtPosAbs)
```

Parameters:

`playerID`: Identifier of the player instance that is the source of the event

`eventCode`: integer value equal to one of the `PE_xxxx` values.

`evtPosAbs`: Absolute position of media stream (in milliseconds) at the time of the event.

Returns: none

ECMAScript Example

8 SOCIAL NETWORKING API GROUP

The Social Networking API Group allows a package to access social networks, such as Twitter, Google+, Facebook, and similar types of on-line communities. The intent is to allow users to share content and/or comments with others. This means that both the posting and receiving of content is supported.

8.1 Overview

This API group is designed on the premise that the Framework is responsible for all interactions with specific social networks. The operational concept is that:

- The Package indicates to the Framework a desire by the consumer to share specific content (e.g. a text comment, an image from the video stream) but does not specify how the content is to be shared nor what social network to use.
- The Framework is responsible for any additional interactions with the consumer that are necessary to resolving which social network is to be used and what the nature of the 'share' is to be.
- The Framework is responsible for any log-in or authorizations procedures specific to the selected social network.
- Any interactions with the social network API is handled by the Framework.

Functionality to be implemented by the Framework is divided into the following subgroups:

Subgroup Summary	
`Basic`	Methods to access and use social networks
`Social Event`	Interface to receive notification of events relating to social networks

A Package will only implement the Account Event subgroup.

8.2 Framework Interface

8.2.1 Sharing Subgroup

Method Summary	
`hasSocialNetworks`	Returns a Boolean indicating if any social networks are available for use
`share`	Initiate a 'share' operation.

8.2.1.1 Events

The following event types may be communicated to a `SocialEventListeners`:

Event	Description
`SN_EVENT_STATE_CHANGE`	Indicates a change in the availability of social networks.
`SN_EVENT_REQ_REJCTED`	Request was rejected by the social network
`SN_EVENT_REQ_PENDING`	Request has not yet been initiated and is still pending
`SN_EVENT_REQ_UNSUPPORTED`	Request not supported in current environment
`SN_EVENT_RCVD_RESP`	A posting has been received in response to and earlier share request
`SN_EVENT_RCVD_UNSOL`	An posting has been received that is unsolicited (i.e., not a response)

8.2.1.2 Constants

The following may be used to indicate the type of content to be shared.

Code	Description
`SN_TYPE_QUARTER`	text
`SN_TYPE_IMAGE`	image
`SN_TYPE_VIDEO`	video
`SN_TYPE_IM`	Short text
`SN_TYPE_CHAT`	Interactive dialog

8.2.1.3 hasSocialNetworks()

Returns a boolean flag indicating if any social networks are available for use in sharing content or postings. The exact meaning of ‘availability’ is determined by the retailer. One criterion might be that a social network is considered available if the Framework has the ability to interact with it at the API level. A more restrictive criterion would be that the current consumer has an established account with the social network. The precise semantics of social network availability should therefore be specified as part of the framework-package integration procedure.

Usage

```
hasSocialNetworks()
```

Parameters: none

Returns:

`true` if any social networks are available, false otherwise

ECMAScript Example

8.2.1.4 share()

Requests the initiating of a 'share' operation. A `contextID` is used to associate this request with any resulting event notifications. The `contentType` parameter indicates the type of content the user intends to share (e.g., text). This is may be used by a Framework implementation to identify the social networks that are compatible with the user's intentions.

An optional `contentUrl` is used to indicate the information that is to be shared. If not provided, the Framework is responsible for initiating and dialogues with the consumer to obtain the information (e.g., providing a pop-up text-input window for entering an instant message). If a `contentUrl` is provided, the specifics of the encoding syntax are outside the scope of this API and should be resolved as part of the Framework-Package integration process.

A share request will be treated as an asynchronous operation and, therefore, does not return a value. Instead, and registered `SocialEventListener` instances will be notified of events relating to the progress of the request.

Usage

```
share(String contextID, int contentType, String contentUrl)
```

Parameters:

`contextID`: String value used in event notifications to identify a share request

`contentType`: integer value equal to one of the `SN_TYPE_xxxx` values

`contentUrl`: String value used to identify the content to share (optional)

Returns: `none`

ECMAScript Example

8.2.2 Social Event Subgroup

Method Summary	
`addListener`	Add a `SocialEventListener` to the listener list.
`removeListener`	Remove the listener from the listener list.

8.2.2.1 addListener()

Adds the listener to the listener list. The listener is registered for all event notifications associated with this interface.

Usage

`addListener(SocialEventListeners listener)`

Parameters:

`listener`

Returns:

`true` if the listener was added successfully, false otherwise.

ECMAScript Example

8.2.2.2 removeListener()

Removes the listener from the listener list. This removes a `SocialEventListeners` that was previously registered for all event notifications.

Usage

`removeListener(SocialEventListeners listener)`

Parameters:

`listener`

Returns:

`true` if the listener was successfully removed, false otherwise.

ECMAScript Example

8.3 Package Interface

Components wishing to receive notification of events related to the use of social networks must register as `SocialEventListeners`. While it is assumed that a Package will implement the `SocialEventListeners` interface, this is not a requirement of the API.

8.3.1 Social Event Subgroup

8.3.1.1 eventNotification()

Notification of an event related to the use of social networks. The `eventCode` value is always provided. The other three parameters are considered optional or required depending on the type of event that is being reported.

- `STATE_CHANGE:` The other three parameters may be ignored. The Listener may choose to invoke the `hasSocialNetworks()` call to determine the current availability of social networks.

- `REQ_xxxx`: The `contextID` will indicate the request that this event is associated with. The remaining two parameters may be ignored.
- `RCVD_RESP`: The `contextID` will indicate a package-initiated request that this event is associated with and `contextID` will indicate the type of response received. The `contentUrl` is used to indicate how the received content may be accessed. The specifics of the encoding syntax are outside the scope of this API and should be resolved as part of the Framework-Package integration process.
- `RCVD_UNSOL`: The parameter interpretation is identical to that associated with the `RCVD_RESP` notification with the one exception of the `contextID`. The `contextID` will be assigned by the Framework rather than used to indicate a previous package-initiated request.

Usage

```
eventNotification(int eventCode, String contextID, int contentType,
           String contentUrl )
```

Parameters:

`eventCode`: integer value equal to one of the `SN_EVENT_xxxx` values.

`contextID`: String value

`contentType`: integer value equal to one of the `SN_TYPE_xxxx` values

`contentUrl`: String value indicating location of received content

Returns: `none`

ECMAScript Example

9 ENHANCEMENTS API GROUP

The Enhancements group addresses features that, while not required, provide support for capabilities that may be used by developers to provide a richer and wider set of interactive experiences. The functionality in this group is considered optional.

9.1 Overview

This group includes functionality that is considered optional. The API therefore provides the package with the ability to query the framework to determine which specific capabilities are provided.

9.2 Framework Interface

The functionality is divided into feature-specific subsets (e.g., support for bookmarks). If a Framework developer chooses to implement an optional feature they must implement the entire subset of functions associated with that feature. Currently the feature subsets are:

Subgroup	Description
Wishlists	Manage wishlists
Bookmarks	Manage bookmarks
History	Persistently store information about the usage of this Package.

9.2.1 Wishlists Subgroup

Wishlist management is outside the scope of this API and some retailers may support the use of multiple wish lists. For this reason the invocation of the `addToWishList` method is to be interpreted as a signal to the retailer framework to initiate whatever processes and user interactions may be necessary to complete the action (e.g., prompting the consumer for which list the content is to be added to).

The assumptions shall be that consumers may add content to a wish list at any time, including during the playback of content. Thus, in the event that some form of interaction with the consumer or a remote site is required, these actions should be handled by the framework in an asynchronous manner. Final results of the operations will be signaled to the requester via an `ListEventListener` notification.

Wishlists are associated with a specific user account. A user must, therefore, be logged in in order for wishlist actions to be processed. It is, however, up to the framework implementer to decide how to handle requests when the user is not currently logged in. The framework may either:

- Reject the request from the package and return a status of `CC_UNSUPPORTED`, or
- Initiate the login process prior to completing the requested operation.

Package implementers should, therefore, allow for either approach.

Subgroup	Method Summary	
Wishlist	`supportsWishList`	Determine if the wish-list capability is provided by the retailer.
	`addToWishList`	Add the specified content to a wish list maintained by the retailer.
	`removeFromWishList`	Remove the specified content from a wish list maintained by the retailer.
	`isInWishList`	Determine if the specified content is in a wish list maintained by the retailer.
	`addListener`	Register a listener to receive ListEvent notifications.
	`removeListener`	Remove a previously registered listener.

9.2.1.1 supportsWishList( )

Determine if the wishlist capability is provided by the retailer. Note that this is an indication of the framework's general ability to provide this capability. It is not an indication that is *currently* able to process wishlist requests as that may be impacted by the user's login status.

Parameters: none

Returns:

`true` – if the wishlist capability is provided by the retailer, false otherwise

ECMAScript Example

9.2.1.2 addToWishList()

Add the specified content to a wishlist maintained by the retailer. Wishlist management is outside the scope of this API and some retailers may support the use of multiple wishlists. For this reason the invocation of the `addToWishList` method is to be interpreted as a signal to the retailer framework to initiate whatever processes and user interactions may be necessary to complete the action. In the event that some form of interaction with the consumer or a remote site is required, these actions should be handled by the framework in an asynchronous manner. The changes to wishlist contents, if any, will be signaled to any registered listeners via a **`ListEvent`** notification.

This function will return a `StatusDescriptor` with a completion code of `CC_OP_INVALID_PARAM` under the following conditions:

- if the specified list already contains the content identified, or
- if the `ListId` is invalid.

If the user is not currently logged in a Framework MAY return a status of `CC_UNSUPPORTED`.

Usage

```
addToWishList(String contentId, String listId, String requestId)
```

Parameters:

`contentId`: - the retailer's content identifier

`listId`: - the wishlist identifier. If `null` then either the default list will be used or the consumer will be queried.

`requestId`: - context value associated with any resulting asynchronous event notifications.

Returns:

`StatusDescriptor` instance with completion code of `CC_OP_COMPLETED` or `CC_OP_INVALID_PARAM`

ECMAScript Example

9.2.1.3 removeFromWishList()

Remove the specified content from a wishlist maintained by the retailer. Wishlist management is outside the scope of this API and some retailers may support the use of multiple wishlists. For this reason the invocation of the `removeFromWishList` method is to be interpreted as a signal to the retailer framework to initiate whatever processes and user interactions may be necessary to complete the action. In the event that some form of interaction with the consumer or a remote site is required, these actions should be handled by the framework in an asynchronous manner.

Final results of the operations will be signaled to the requester via an `ListEvent` notification. If the content is not currently present on the specified wishlist, the function will return a status of `CC_OP_COMPLETED` but no event will be generated since the list was not modified. This function will return a `StatusDescriptor` with a completion code of `CC_OP_INVALID_PARAM` if the `ListId` is invalid. A framework MAY return a status of `CC_UNSUPPORTED` if the user is not currently logged in

Usage

```
removeFromWishList(String contented, String listId, String requestId)
```

Parameters:

`contentId`: - the retailer's content identifier

`listId`: - the wishlist identifier. If `null` then either the default list will be used or the consumer will be queried.

`requestId`: - context value associated with any resulting asynchronous event notifications.

Returns:

`StatusDescriptor` instance

ECMAScript Example

9.2.1.4 isInWishList()

Determine if the specified content is in a wish list maintained by the retailer. Retailers may choose to support the use of multiple wishlists. The value passed to the callback function is, therefore, a list identifier. In the event that the content is not found on any list or if the request cannot be processed due to the user not being logged in, a `null` value is returned.

Usage

```
isInWishList(String contentId, String requestId, Function callback)
```

Parameters:

`contentId`: - the retailer's content identifier

`requestId`: returned to callback function with final result

`callback`: package function to receive result.

Returns:

`StatusDescriptor` instance

Callback

```
function(String requestId, String listId)
```

Arguments:

`listId` – if the content is in a wishlist, `null` otherwise

ECMAScript Example

9.2.1.5 addListener()

Adds the listener to the listener list. The listener is registered for all event notifications associated with this interface.

Usage

```
addListener(ListEventListener listener)
```

Parameters:

`listener`

Returns:

`true` if the listener was added successfully

ECMAScript Example

9.2.1.6 removeListener()

Removes the listener from the listener list. This removes a `ListEventListener` that was previously registered for all event notifications.

Usage

```
removeListener(ListEventListener listener)
```

Parameters:

`listener`

Returns:

`true` if the listener was successfully removed

ECMAScript Example

9.2.1.7 List Event Notifications

Additions or deletions from a wishlist will result in the generation of a `ListEvent` that will be delivered to all registered listeners. Changes to the contents of a list may be due to a request from the consumer (e.g., a purchase) or may result from some event or action not initiated by the consumer. If the change is derived from a consumer initiated request, any `requestId` provided when the request was made will be included in the event notification. The notification will also identify which list has been changed.

Usage

```
eventNotification(String contentId, String listId, String requestId,
          int eventCode)
```

Parameters:

`contentId`: the content identifier

`listId`: the wishlist identifier.

`requestId`: identifier associated with the request that resulted in the change to availability status (optional)

`eventCode`: the applicable code `[ LE_ADDED | LE_DELETED ]`

Returns: none

ECMAScript Example

9.2.2 Bookmarks Subgroup

A 'bookmark' is specified as a location within a media stream. It is therefore defined by two values: a contentId, indicating a media stream, and a position, given in milliseconds.

The actual bookmark position should be assumed to be the frame at exactly that time or immediately following. Frame accuracy should not assumed for bookmarks. The exact position of a bookmark depends on the encoding of the content. Changes in frame rate and time compression can affect bookmark position. See Section 9.2.2.1 for further guidance on the implementation and usage of bookmarks.

Subgroup	Method Summary	
Bookmarks	`supportsBookmarks`	Determine if the bookmark capability is provided by the retailer.
	`setBookmark`	Set a bookmark at the designated location in the media.
	`removeBookmark`	Remove any bookmarks at the designated location in the media
	`getBookmark`	Returns the bookmark properties as a set of key/value pairs.
	`getAllBookmarks`	Returns sorted array of locations in the media for which a bookmark exists.

9.2.2.1 Use of Timecodes

A 'bookmark' is specified as a location within a media stream and is defined by two values: a contentId, indicating a media stream, and a position, also referred to as a *timecode*, given in milliseconds. The actual bookmark position should be assumed to be the frame at exactly that time or immediately following. Frame accuracy should not be assumed for bookmarks.

The exact frame that equates a bookmark's position depends on several factors. Changes in encoding, i.e., frame rate or time compression, can affect bookmark position in that a position specified in terms of milliseconds will now map to a different frame. Furthermore, any change in the specific edit (e.g., theatrical vs. director's cut) will also affect bookmarks. Editing or encoding changes will, however, also result in the assignment of a new contentId. The ability to define bookmarks applicable to a set of related content (e.g., both a theatrical and director's cut of the same movie) would require the ability to 'translate' a bookmark in terms of identifying the equivalent frame (i.e., time-based offset) in each version of the content. This type of advanced capability is outside the scope of this API.

To avoid bookmark collisions, a timecode specified for use as a bookmark position SHALL be assumed to have a resolution of ½ second. Any two bookmarks that are defined with the same contentId and that have positions within ½ second of each other SHALL be treated as equivalent and redundant. See Section 9.2.2.3.2 for additional details regarding the handling of duplicate bookmarks.

9.2.2.2 Bookmark Properties and Structure

This section of the API should be considered a work-in-progress and likely to change in future releases.

A bookmark SHALL have as a minimum the following required properties:

- `cid` – identifies a specific media stream
- `tstamp` – offset from start of stream (in milliseconds)
- `label` – user entered string

A bookmark MAY have the following optional properties:

- `description` - user entered string
- `created` – Date and time of bookmark creation in JSON-compatible format (see below)
- `lastAccess` - Date and time the bookmark was last selected by the user (JSON-compatible format)
- `selectCnt` – Number of times the user has selected the bookmark

A framework implementer MAY provide additional properties provided these properties are considered optional and the property names do not conflict with those defined by this API.

A bookmark SHALL be compatible with the JSON syntax. This means that all date-time fields are encoded as strings using UTC times (e.g., `2015-12-05T21:01:26.938Z`)

9.2.2.3 Framework Interface

9.2.2.3.1 supportsBookMarks( )

Determine if the bookmark capability is provided by the retailer.

Usage

```
supportsBookMarks()
```

Parameters: none

Returns:

`true` – if the bookmark capability is provided by the retailer, false otherwise

ECMAScript Example

9.2.2.3.2 setBookmark()

Set a bookmark at the designated location in the media. A bookmark will be rejected if the `contentId` is unrecognized or if it duplicates an already existing bookmark. Duplication is determined by the comparison of `contentId` and `position`. See Section 9.2.2.1 for further details.

Usage

```
setBookmark(String contentId, int position, String label, String
          description, Function callback)
```

Parameters:

`contentId`: - the retailer's content identifier

`position` absolute position in time given in milliseconds

`label`: identifier to display to consumer

`description`: optional text

`callback`: function to receive final result

Returns:

`true` – if the bookmark has been added, false otherwise

Callback

```
function(Object bookmark)
```

Arguments:

`bookmark` – the added bookmark formatted as a JSON-compatible object. See Section 9.2.2.2.

ECMAScript Example

9.2.2.3.3 removeBookmark()

Remove any bookmarks at the designated location in the media.

Usage

```
removeBookmark(String contentId, int position)
```

Parameters:

`contentId`: - the retailer's content identifier

`int position`: absolute position in time given in milliseconds

Returns:

`true` – if a bookmark has been removed, false otherwise

ECMAScript Example

9.2.2.3.4 getAllBookmarks()

Returns an array of bookmarks, each formatted in accordance with Section 9.2.2.2.. If no bookmarks exist int the specified content an empty array is returned.

Usage

```
getAllBookmarks(String contentId)
```

Parameters:

`contentId`: - the retailer's content identifier

Returns:

`Object[]` array

ECMAScript Example

9.2.2.3.5 getBookmark ()

Returns a bookmark formatted in accordance with Section 9.2.2.2. If no bookmark exists at the specified position a `null` value is returned.

Usage

```
getBookmark (String contentId, int position)
```

Parameters:

`contentId`: - the retailer's content identifier

`position:` absolute position in time given in milliseconds

Returns:

Object

ECMAScript Example

9.2.2.3.6 getRelativeTo ()

Returns the bookmark closest to the specified location when transitioning in the indicated direction. The bookmark is formatted in accordance with Section 9.2.2.2. If no bookmark exists that satisfies the criteria a `null` value is returned.

Usage

```
getBookmark (String contentId, int position, Boolean fwdDir)
```

Parameters:

`contentId`: - the retailer's content identifier

`position`: absolute position in time given in milliseconds

`fwdDir`: if `true` the returned bookmark should have a timecode greater than the that specified by the `position` argument. If `false` the timecode must be less than that of the `position` argument.

Returns:

Object

ECMAScript Example

9.2.3 Package History Subgroup

These functions provide a mechanism for a package to persist information regarding how a specific user has made use of the package.

A package history may include user preferences specific to the package but it may also include viewing history such as which clips have been viewed or the point in a clip at which the user last paused and exited the package. Thus, the package history may be used to pause the interactive experience, persist its state, and then resume the experience at a later time.

Any information that is saved persistently will be treated by the Framework as a text blob. The content and structure are specific to the package. The formatting MUST be JSON compatible. See Section A.4.3 for further guidance on this topic.

The location and mechanism for persisting package history is determined by the retailer. No assumption is made by this API as to whether history data is specific to the current Viewing Environment or if a retailer is maintaining history data on a global (i.e., cross-platform) basis.

Subgroup	Method Summary	
PackageHistory	`supportsHistory`	Determine if the package history capability is provided by the retailer.
	`getPackageHistory`	Request for information regarding the previous use of the package by the currently signed-in consumer.
	`savePackageHistory`	Persist information regarding the latest use of the package by the current consumer.

9.2.3.1 supportsHistory( )

Determine if the package history capability is provided by the retailer.

Usage

```
supportsHistory()
```

Parameters: none

Returns:

`true` – if the capability is provided by the retailer, false otherwise

ECMAScript Example

9.2.3.2 getPackageHistory()

Request for information regarding the previous use of the package by the currently signed-in consumer. The history is provided in the format of a JSON-compatible object that is returned asynchronously to the caller via a callback.

Usage

```
getPackageHistory(Function callback)
```

Parameters:

`callback`: function to receive final result

Returns:

`StatusDescriptor` instance indicating `CC_OP_IN_PROGRESS`, `CC_OP_COMPLETED`, or `CC_OP_UNSUPPORTED`

Callback

```
function(Object history)
```

Arguments:

`history` – the package history formatted as a JSON-compatible object.

ECMAScript Example

9.2.3.3 savePackageHistory()

Request to persistently store information regarding the latest use of the package by the current consumer.

Usage

```
savePackageHistory(Object history)
```

Parameters:

`history` – the package history formatted as a JSON-compatible object .

Returns:

`StatusDescriptor` instance

ECMAScript Example

ANNEX A. IMPLEMENTATION GUIDANCE

The contents of the section are for informative purposes only and are intended to provide guidance to developers and to assist in their use and understanding of the normative material.

A.1. Concept Overview

The motivation and background context of the Cross-Platforms Extras project can be viewed in terms of two viewpoints: the end-goals and the immediate goals.

The *end-goal* of this effort is to provide home users with enhanced interactive features that extend beyond simply watching a movie to include support for activities such as social interactions, access to supplementary information, and commerce. Examples of supported actions may include accessing behind-the-scenes commentary, posting comments on social media, buying or renting a film, or adding a film to a wish-list for later purchase.

The *immediate goal* of this effort is to facilitate the coordinated efforts of both content producers and content retailers/distributors in the creation of these packages. In addressing this goal, there is a requirement that the solution provides support for any/all retailer distributing content from any/all content provider. To that end an API is required, along with any supporting contextual material that assists interested parties in the creation and deploy of these capabilities.

A.1.1. The Consumer's Experience

- The package has to operate in a reliable and user-friendly manner regardless of where the consumer is and what type of device they are using. That means that as part of its initialization process a package will need to determine, and adapt to, device-specific capabilities such as the device's screen size, and the capabilities of any available network connection, and the availability of local storage for downloading and caching.
- More often than not, consumer will be viewing content on mobile devices. This means that even after start-up and initialization, a package may be presented with a wide range of events to handle.
  - The screen has changed orientation and size
  - The quality of the network connection has changed (e.g., lower bandwidth, intermittent loss of connectivity, longer latencies)
  - The possibility of new types of interrupt events (e.g., incoming phone call pre-empts playback).
  - A warning or forced shutdown due to low battery power

Regardless of the nature of change or preemption, the consumer expects the viewing experience to continue uninterrupted or to resume where they left off

- Consumers may start watching a movie late at night at home on a desktop PC, pause it part way thru, then resume watching the next morning using their smartphone while commuting to work on the train. The retailer may be the same but the framework they provide may be different on a mobile device than on a desktop one (e.g., different browsers; same browser but different media players; pure HTML browser on the desktop but an app on the mobile device). Regardless of the change in environment, the consumer expects the viewing experience to have a similar enough experience in terms of functionality that they don't feel overly inconvenienced by being 'mobile'

A key goal of this effort is to provide an architecture and API that will support the expectations of all relevant parties in terms of access to a modern and full-featured viewing experience. Those expectations include:

- Viewing has become a **content-centric social experience**: The consumer may be watching content by themselves or with friends and family in the same room. Regardless, they will be offered the opportunity to simultaneously engage with cast members and/or other consumers interested in the same content via a variety of social networking channels. Possible on-line communities and networks include, but are not limited to, (in alphabetical order) Facebook, Flickr, Flixster, Google+, Instagram, Orkut, Pinterest, Sina Weibo, Twitter, or Tumblr. Consumers will be able to post, comment, friend, follow, vote, enter a contest, or play a game at the same time they are viewing.
- Viewing is a **content-centric shopping opportunity**: While the consumer is watching a movie or show, content providers and retailers may wish to provide the opportunity to purchase related merchandise such as clothes, games, jewelry, posters, music, and books. The items being displayed may change during playback so as to relate to the scene currently viewed (e.g., the consumer can purchase and download the song currently being played in the soundtrack or buy a dress identical to that being worn by an actress).
- Viewing may be a **multi-threaded experience**: The consumer may, or may not, pause the video playback while making use of a social-networking or retail shopping component. Use of these ancillary components might potentially involve graphic overlays on the primary screen, the temporary display of a secondary screen, and/or a change to the size of the primary screen.
- Viewing may be a **tailored personalized experience**: by using profiles and analytic data, either gathered by the packages or from some external source, the consumer's viewing experience and interactions may be custom tailored. This may range from relatively simple preference-based selections (e.g., which language is used in text displays) to more advanced forms of on-the-fly customization such as the display of available merchandise based on the analysis of past in-package purchases.

A.1.2. Concept of Operations

In identifying the functions and constructs that are within the scope of the Cross-Platform Extras API, an underlying concept of operation is assumed. This concept assumes a division of responsibility between a Framework and a Package. The key points are:

- The functions and responsibilities of the Framework are only those that are necessary and sufficient to enable a package to perform its functions.

- The Framework is responsible all user-specific (i.e., account) interactions and functionality, instantiation of a player, and launching a Package. Retailers provide an environment (a.k.a. the Framework) that allows the consumer to browse the content available for purchase or rental, add selected content to their private or shared collection, access (i.e., play) content in their collection, and pause and save the state of playback, returning to it at a later time and, possibly, on a different device.
- The Package is responsible media selection and control. It is, while in the active state, responsible for the layout of the user interface any responding to any event related to either a user action or a condition associated with the hardware environment. When the consumer selects specific content for playback, the retailer's framework fires off a 'package' that is intended for the presentation and viewing of that specific content (i.e., movie, TV show, etc.) in that specific retail framework. The package defines its own UI (including look-and-feel) and any supplementary content and interactions above and beyond simply playing the primary content (i.e., the movie). When a package is in the RUNNING state it has responsibility for, and control of, the user experience.
- Packages developers may, if they choose, implement additional capabilities outside the scope of this API. In doing so they are free to directly access native API to obtain additional information or capabilities (e.g., call status on a mobile device). If, however, developers choose to implement this type of capability, it must be done in a manner that does not conflict with the state behaviors specified by this API.

The remainder of this document is intended to define an architecture and API that will support the defined concept of operation where support is needed and not hinder it everywhere else.

A.1.3. Concept of Deployment

A key requirement is that the CPE technologies not impose any limitations on the ability of any Retailer to provide a distribution channel for any Content Provider. Neither should it impose any limitations on the nature of the supplementary material a Content Provider chooses to make available to Consumers.

The user interface presented to a consumer at any given moment will be a combination of components and resources provided by both the Retailer and whichever Content Provider has ownership of the package currently being presented to the Consumer. During any period in which a package is *not* being presented, the UI shown will be a Retailer-specific default. All interactions between a framework and package will comply with the API specified in this document. This does not, however, preclude or prohibit additional interactions between the software and systems of a Retailer and that of a Content Provider.

The division of responsibilities between the Retailer and Content Provider in terms of providing UI components, resources, and behaviors are as follows:

- Retailer / framework:
    - overall style, layout, and look-and-feel of the UI prior to content selection or after the user has terminated an Interactive Experience (i.e., while the framework is in the non-CPEP state)

    - choice of player
    - responding to all user interactions whenever there is no package in the Running state (e.g., log-in, setting of preferences, account management)
    - responding to package requests while an package is in the Running state
    - inclusion of any content specific to that Retailer (e.g., a 30 second advertisement played before the movie)
    - ensuring a clean environment upon start-up of an package (i.e., garbage collection)
- Content Provider / package:
    - providing primary content (i.e., a movie or TV show)
    - providing all Extras (i.e., secondary content) such as deleted scenes, interviews with actors, directory's commentary, etc.
    - responding to all user interactions while the package is in the Running state

This behavior is consistent with the state machine specified in Section 4.1.1: Package Lifecycle. The nature and behavior of the "interactive experience" presented to the consumer will be determined by the inner state of the package's `Running` state. This will, however, be transparent to the framework.

A.2. Design Principles

The Cross-Platform Extras API has been developed with the goal of supporting the operational concepts discussed in Section A.1. It is recognized, however, that developers may wish to extend and enhance the 'core' capabilities, that goals and concepts will evolve over time, and that the capabilities and components provided by the viewing environments will also evolve. The flexibility, longevity, and extensibility of both the API and any conformant products is, therefore, a paramount concern.

The remainder of the section identifies the design principals by which the Cross Platforms Extra API addresses these concerns.

A.2.1. Modular Object Oriented Design

The API is defined in terms of four functional groups (see Section 3.1).The methods and capabilities within each grouping are defined based on object-oriented design principals. This does not, however, mandate the use of an object-oriented language (i.e., Java, C++, etc.). Neither is any restriction or assumption made as to how developers choose to assign functional groups to components. The only assumption made is that there is a set of one or more software modules that provide the 'framework' functionality and another set of one or more modules that provide the 'package' functionality.

Retailer frameworks are to be agnostic and flexible in terms of support of packages. That is, packages from different content providers may be swapped in and out of the framework at will. To ensure this sort of plug-n-play capability the API is based on the use of zero-argument constructors. See Section 3.3.1 for further details.

A.2.2. Support for Variation in Player Capabilities

The standards and capabilities of media players are of critical importance to this effort. It is also an area undergoing rapid evolution. Emerging technologies include, but are not limited to, HTML5, Media Source Extensions (MSE), Encrypted Media Extensions (EME), H.265 and WebM. Player implementations may vary in the options and capabilities supported or in the manner of configuring. The CP Extra API is intended to allow operation in a wide range of common viewing environments. In API accomplishes this goal by abstracting common properties or features, and support the union of all appropriate features.

The API also groups related features and allows some groups to be optional. That allows Framework implementers to onboard more quickly and provides some environment flexibility.

A.2.3. Mobile Users

Support for mobile devices is a critical requirement as the use of mobile devices is increasingly becoming the primary mode of on-line activity. A viewing environment based on a mobile platform has several characteristics that must be allowed for. First and foremost is that any application running in a mobile OS must be able to cleanly handle suspension and resumption due to preemption due to incoming calls. Second is the need to deal with the variable and intermittent nature of mobile networks. Third is the need to handle events not encountered when operating in a desktop environment such as a change in screen layout due to reorientation of the device or warnings of low battery power.

It is the responsibility of whichever component has active control of the user interface to respond to any of these events. That is to say, that responding to events is the responsibility of the Package when the package is in an active running state and that at all other times it is the responsibility of the framework.

No requirement is imposed that a either type of component be designed to respond or take into account any of the conditions identified above. It is expected, however, that component developers, especially those focused on supporting mobile viewing environments, will find it advantageous to do so. The API provides methods and data structures for obtaining information regarding the current state of network connectivity (Sections 4.3.1.5 and 4.4.1) and receiving asynchronous notification of events relating to the device (Section 4.3.3). Component developers may also choose to access native API in order to obtain additional data. See Section B.2 for further discussion.

A.3. Guidance for Framework Developers

A.3.1. Functional Decomposition

This API decomposes the functionality into four groups: Package Management, Content Access, Account Access, and Player Interaction. Framework implementers are responsible for providing all four. There is, however, neither requirement nor assumption as to how many software components are used to implement the functionality. Framework developers may choose to provide a single construct (e.g., a JavaScript file) or split the functionality across multiple constructs for purposes of modularity, performance, or flexibility.

A.3.2. Package Management

The Framework has ultimate responsibility for ensuring that the software providing the Interactive Experience is stable and well-behaved. At the same time, Framework developers have no guarantee that a deployed Package is entirely bug free or can properly respond to all external events. The Framework should, therefore, take care to ensure that, as the "last resort" in case of a fatal error, any allocated resources are released and a final garbage collection and clean-up is performed.

Framework developers may also API to monitor the state-behavior of a Package and insure that it is still alive and function. This may be done via the Package Management API group's `getState()` method (see Section 4.3.1.5)

A.4. Guidance for Package Developers

A.4.1. Mobile Users

Any viewing environment based on a mobile platform has several characteristics that must be allowed for. First and foremost is that any application running in a mobile OS must be able to cleanly handle suspension and resumption caused by preemption due to an incoming call. Second is the need to deal with the variable and intermittent nature of mobile networks. Third is the need to handle events not encountered when operating in a desktop environment such as a change in screen layout due to reorientation of the device or warnings of low battery power.

Assignment of responsibilities for the correct handling of these situations is outside the scope of this API. Package developers should, therefore, not assume that the Framework will insure correct behavior. The roles and responsibilities of each component in this regard should be identified as part of the integration process. Access to these events is outside the scope of this API and the mechanism for accessing may depend on the viewing environment (e.g., iOS vs. Android), the implementation language (e.g., Java vs. JavaScript) or both. See Section B.2: "Mobile Environments" for additional discussion of these issues.

A.4.2. Single User with Multiple Devices

The use of multiple devices by a single individual is an increasing trend. A user may begin watching a movie on a tablet device while coming home from work on the train, then finish watching after dinner on a desktop PC. The ability to pause an activity on one device and seamlessly resume on another is referred to *global session persistence*. This has become a characteristic that consumers expect to find supported. In contrast, *local session persistence* only allows a user to resume an activity on the device it was previously initiated on.

The functions of the Cross Platform API that address session persistence are incorporated in the Enhancements API Group (see Sections 9.2.2 and 9.2.3). The API, however, does not differentiate between the global and local forms of session persistence. Support of either form is not required to be considered in compliance with this specification.

In the event a Retailer maintains session persistence information, the Cross-Platform Extras API defines methods to allow a package to communicate to the framework those aspects of the session state it wishes to persist and that when started it may obtain from the framework the data necessary to (a) determine that a previous session state exists and (b) retrieve and restore the paused session. Whether or not it may do so on a global or local basis will be determined by the capabilities the retailer chooses to offer via their framework implementation.

The back-end services and infrastructure used to provide global session persistence are outside the scope of this API. Each content provider will be free to decide if, and how, a Package implementation will support this type of functionality. If session persistence is offered to consumers, either on a global or local basis, package developers may choose to either implement the capability using whatever functionality the Framework provides via the Enhancements API, or implement an "organic" capability that is built upon back-end services and infrastructure that are provided by the content provider. Whether or not a Package uses this in lieu of, or in combination with, its own persistence mechanism is outside the scope of this API.

A.4.3. History Data

The package history may be used to when the interactive experience is paused so that when it resume, it continues where it left off. Package history data may include user preferences specific to the package but it may also include viewing history such as which clips have been viewed, or the point in a clip at which the user last paused and exited the package.

ANNEX B. ADAPTATION TO SPECIFIC VIEWING ENVIRONMENTS

The contents of the section are for informative purposes only and are intended to provide guidance to developers and to assist in their understanding and use of the normative material.

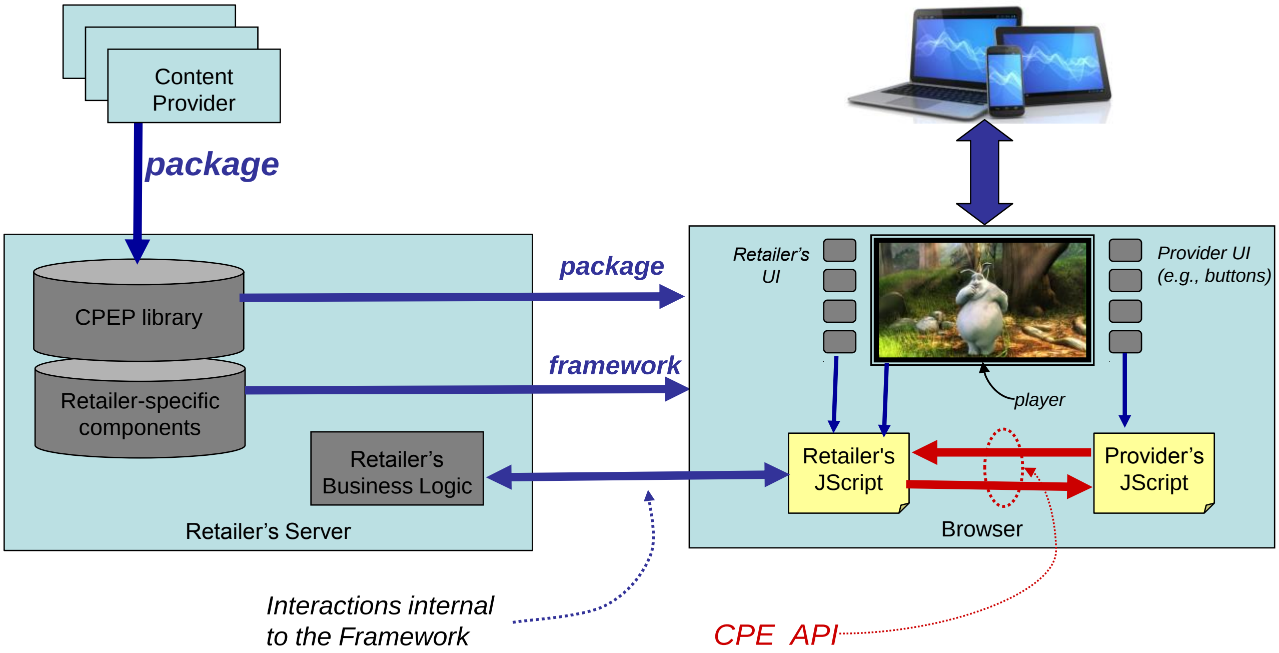
B.1. HTML5

An HTML5 implementation will use the consumer's browser as the viewing environment. The Package will, therefore, be implemented in JavaScript and use CSS and standard HTML5 tags to create the viewing experience. When implementing a complete CP Extras experience in this type of environment, developers will have multiple options in terms of where various components reside. These include:

- "All in Browser" approach: Framework contained in HTML page with the Package. The CP Extras API is, therefore, implemented via JavaScript calls.
- "Back-End Framework": Framework resides primarily on server and CP Extras API is implemented via HTTP and AJAX.

B.1.1. All in Browser

An "All in Browser" approach is one in which the user is provided with web pages that contains either only the retailer's Framework (prior to Package selection) or both the Package and the Framework (after Package selection). The CP Extras API is, therefore, implemented via JavaScript calls.

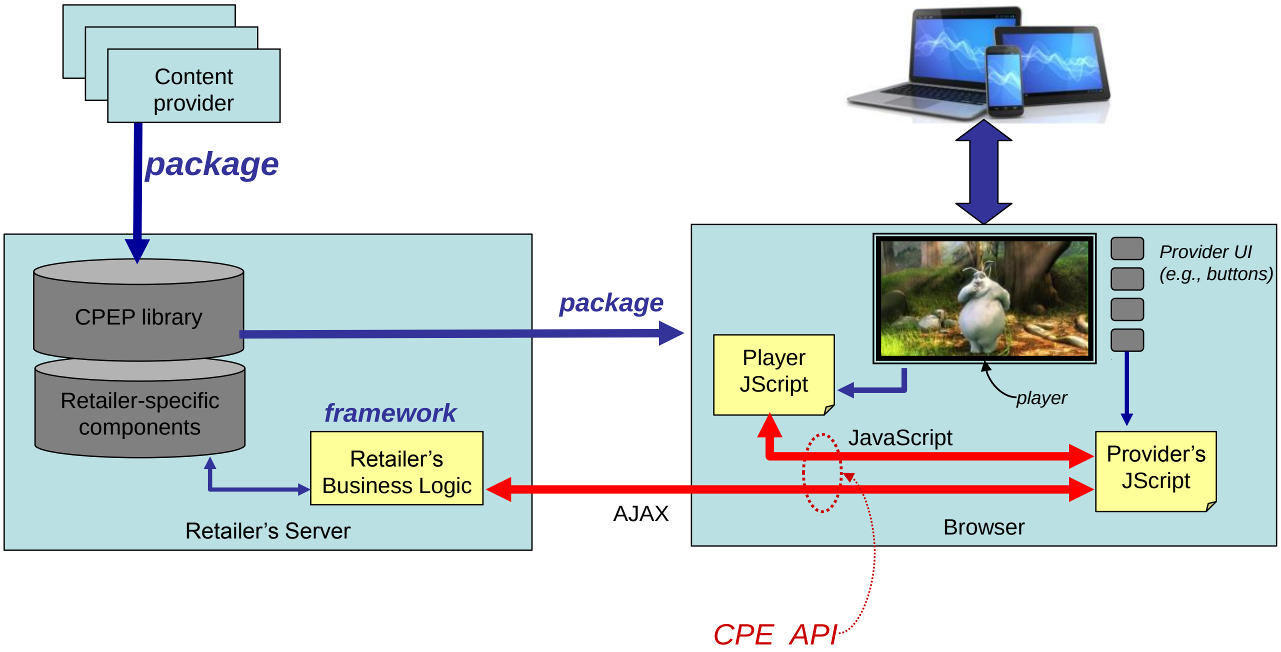


This approach, as illustrated above, begins with a Framework-only HTML5 page being loaded into the browser. Selection of a specific experience by the consumer would result in the Framework

JavaScript using the HTTP protocol to download the appropriate Package, insert it into the HTML page's DOM structure, and then invoking the Package's `initialize` method (see Section 4.3.1)

B.1.2. Back-End Framework

An "Back-End Framework" approach is one in which the user is provided with web pages that contains either only the retailer's Framework (prior to Package selection) or only the Package and Player (after Package selection). The CP Extras API is, therefore, implemented primarily via AJAX interactions.



This approach, as illustrated above, begins with a Framework-only HTML5 page being loaded into the browser. Selection of a specific experience by the consumer would result in the Framework providing the browser with a web page that contains a media player, possible with a supporting JavaScript library, along with the selected Package's JavaScript and DOM structure.

A key aspect of this implementation approach is that it takes advantage of the CP Extra API groups. Framework functionality supporting the Package Management, Content Access, and other API groups resides on the server and therefore is accessed via a binding of the API to AJAX. The one exception is the Player Interaction API group. For performance and flexibility reasons, this group is supported via a JavaScript library loaded with the web page. Player API interactions are therefore supported via JavaScript function calls. An HTML `onload` event may be used to trigger invoking the Package's `initialize` method.

B.2. Mobile Environments

Developers may wish to provide implementations designed specifically for one or more families of mobile devices. Several approaches may be adopted including use of a cross-platform mobile web-based application framework such as PhoneGap or Titanium, or “going native” and directly accessing the iOS or Android API.

B.2.1. Mobile Web-Based Frameworks

There are a large number of cross-platform development frameworks and providing guidance specific to any one of these is outside the scope of this document. A significant number of these are based on the use of HTML5, CSS, and JavaScript. These are referred to as *mobile web-based frameworks*. The primary value-added of these frameworks is access to native APIs unique to mobile devices (e.g., location services) and integration into the mobile OS environment (e.g., application manifests). Due to their usage of HTML5, CSS, and JavaScript, this type of framework may provide a relatively easy migration path from the pure-browser approach discussed in Section B.1.

B.2.2. Native APIs

This section addresses the implementation of Cross Platform Experiences using the native operating environments of a mobile device. The primary challenge that will face a development team is determining how to implement a Framework that provides the dynamic package launching capability that is a core aspect of the Cross Platform Experience API. There remainder of this section examines this issue for two of the most prevalent mobile OS environments. The material provided is not an exhaustive analysis of the issue. Neither should it be regarded as a normative component of the API.

B.2.2.1. Android

The API defined in this document may be easily mapped to an Android application. The Framework may be implemented as a complete application, including the manifest and the main Activity and, potentially, additional activities, services, and content providers. The Framework side of all API groups specified in this document would then be defined as interfaces to be implemented by these components.

A key characteristic of the Cross Platform Extras concept is that Packages may be dynamically loaded. There are several ways this capability may be provided and the material in this section is not intended to be a complete listing of all suitable designs.

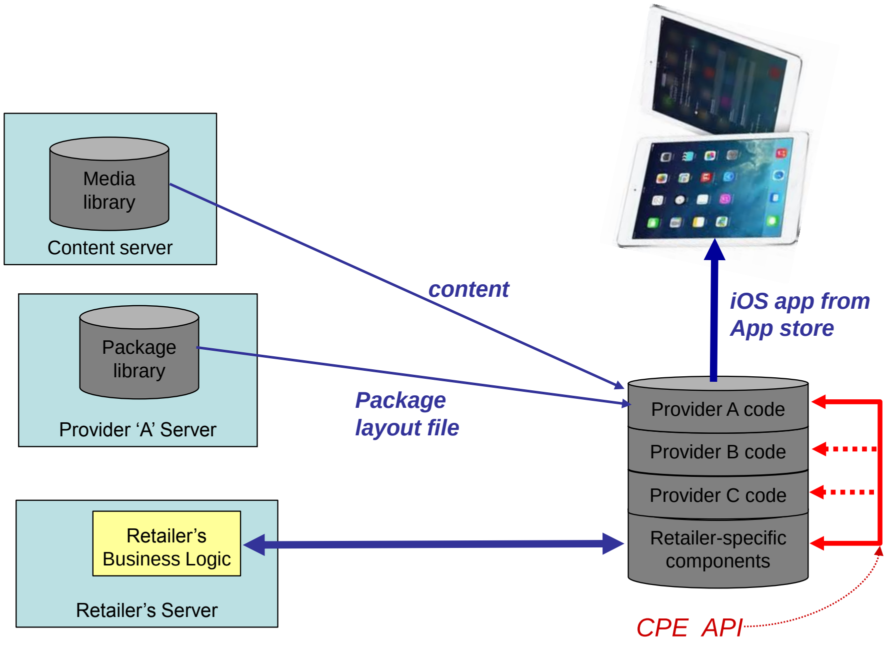
One option is to deploy the Android application with a manifest and class file that includes an Android service for each content provider supported by the retailer. This service would be responsible for (a) implementing all aspects of the Package side of the API and (b) configuring the layout for a given package. Package-specific layouts would take the form of XML and or JSON files accessed either from local storage or over the network from the appropriate server. In other words, a content provider’s Java code may be viewed as a *package template* that (a) implements all aspects of this API document and (b) tailor’s it user interface based on the dynamically loaded *package layout* file (i.e., the XML/JSON).

Another option would be to dynamically deploy the entire package, including any supporting background services or fragments. The Dalvik VM provides facilities for an Android application to perform custom class loading. Instead of loading a Dalvik executable (“dex”) files from the default location, an application can load them from alternative locations. This includes the ability to access and load dex files over the network.

B.2.2.2. iOS

The iOS environment provides limited alternatives for implementing the type of dynamic package-launching capability envisioned by the Cross Platform Extras concept. Specifically, the ability to dynamically download Objective-C code and incorporate it into the app at runtime is not supported. One solution, therefore, is that applications be deployed with both the framework code and package-specific code for each supported content provider (i.e., studio). Tailoring of the UI and behavior to a specific film (i.e., package) could be handled via the accessing and loading of XML or JSON files. In other words, a content provider's Objective-C code may be viewed as a *package template* that (a) implements all aspects of this API document and (b) tailor's it user interface based on the dynamically loaded *package layout* file (i.e., the XML/JSON).

The concept, along with one possible instantiation, is illustrated in the following figure:



The application itself would be downloaded and installed from the Apple App store. In this example, three content providers are supported by the retailer's app. At any given time, the retailer's Framework will interact with at most one of the provider modules via the CP-Extras API. This will be determined by which provider is responsible for the Experience selected by the consumer. Note that the mechanism by which the retailer framework indicates to the package template which specific package has been selected (e.g., "Big Buck Bunny: the Directors Cut") would be via the context parameter used when invoking the package initialization method (see Section 4.3.1).

ANNEX C. EXAMPLES

Full examples can be found at http://test.movielabs.com/cpe.