

CableLabs® Guidelines

Cable Application Developer Guidelines

CL-GL-DG-V01-100205

RELEASED

Notice

This OpenCable guideline is the result of a cooperative effort undertaken at the direction of Cable Television Laboratories, Inc. for the benefit of the cable industry and its customers. This document may contain references to other documents not owned or controlled by CableLabs. Use and understanding of this document may require access to such other documents. Designing, manufacturing, distributing, using, selling, or servicing products, or providing services, based on this document may require intellectual property licenses from third parties for technology referenced in this document.

Neither CableLabs nor any member company is responsible to any party for any liability of any nature whatsoever resulting from or arising out of use or reliance upon this document, or any document referenced herein. This document is furnished on an "AS IS" basis and neither CableLabs nor its members provides any representation or warranty, express or implied, regarding the accuracy, completeness, noninfringement, or fitness for a particular purpose of this document, or any document referenced herein.

© Copyright 2010 Cable Television Laboratories, Inc.
All rights reserved.

Document Status Sheet

Document Control Number:	CL-GL-DG-V01-100205			
Document Title:	Cable Application Developer Guidelines			
Revision History:	V01 - 2/05/10			
Date:	February 5, 2010			
Status:	Work in Progress	Draft	Released	Closed
Distribution Restrictions:	Author Only	GL/Member	GL/Member/Vendor	Public

Trademarks:

CableLabs®, DOCSIS®, EuroDOCSIS™, eDOCSIS™, M-CMTS™, PacketCable™, EuroPacketCable™, PCMM™, CableHome®, CableOffice™, OpenCable™, OCAP™, CableCARD™, M-Card™, DCAS™, tru2way™, and CablePC™ are trademarks of Cable Television Laboratories, Inc.

Contents

1	SCOPE	1
1.1	INTRODUCTION AND OVERVIEW	1
1.2	PURPOSE OF DOCUMENT	1
2	REFERENCES	2
2.1	INFORMATIVE REFERENCES	2
2.2	REFERENCE ACQUISITION	2
3	TERMS AND DEFINITIONS	3
4	ABBREVIATIONS AND ACRONYMS	4
5	CONVENTIONS	5
5.1	SPECIFICATION LANGUAGE	5
6	WBA APPLICATION ENVIRONMENT	6
6.1	LEGACY CABLE PLATFORM APPLICATION ENVIRONMENT	6
6.1.1	<i>Legacy Cable Platforms</i>	6
6.1.2	<i>Memory and Application Size Constraints</i>	6
6.1.3	<i>Concurrent Application Execution</i>	7
6.1.4	<i>Color Space Considerations</i>	7
6.1.5	<i>Resolution Considerations</i>	7
6.1.6	<i>Remote Control Key Event Considerations</i>	7
6.1.7	<i>Image Processing</i>	7
6.2	OCAP APPLICATION ENVIRONMENT	9
6.2.1	<i>Service Provider Applications</i>	9
6.2.2	<i>Service Provider APIs</i>	10
6.3	APPLICATION LIFECYCLE GUIDELINES	11
6.3.1	<i>OCAP Application Lifecycle</i>	11
6.3.2	<i>Bound Application Prompts</i>	14
6.4	WINDOW AND FOCUS MANAGEMENT GUIDELINES	14
6.4.1	<i>Window and Focus Management Model</i>	14
6.4.2	<i>Window and Focus Management on OCAP Platforms</i>	14
6.4.3	<i>User Input Model</i>	16
6.5	GRAPHICS AND DRAWING GUIDELINES	18
6.5.1	<i>HScene Clipping</i>	18
6.5.2	<i>Repairing Overlays</i>	19
6.5.3	<i>Active Rendering</i>	20
6.6	PIXEL BURN-IN	20
6.7	SHARED VIDEO GUIDELINES	20
6.7.1	<i>Shared Video Proxy</i>	21
6.7.2	<i>Shared Video Handler</i>	22
6.7.3	<i>OCAP Shared Video Functionality</i>	23

Figures

FIGURE 1 - SERVICE PROVIDER APPLICATIONS AND REGISTERED APIS10
FIGURE 2 - OCSCENE15
FIGURE 3 - OCSCENECONTROLLER.....16
FIGURE 4 - FULLY BLENDED SCENES19
FIGURE 5 - FULLY BLENDED SCENES OVER VIDEO19
FIGURE 6 - OCsharedvideomanager21

Tables

TABLE 1 - PNG SUPPORT IN ETV.....8

1 SCOPE

1.1 Introduction and Overview

This document presents a set of guidelines and conventions for the development and deployment of well-behaved *cable applications*. Well-behaved applications, by adhering to these guidelines, will create a consistent and recognizable user experience (UE) for interactive television viewers. While these guidelines are voluntary, all application developers are strongly encouraged to embrace them in order to provide all viewers with a familiar and intuitive user experience, regardless of the specific service they are using.

This effort originated from the desire to define a cable industry convention to announce and launch television enhancements. It quickly became clear that program enhancements were only a part of the picture, and that a more complete set of UE guidelines would be required, addressing all types of applications, both program-related and non-program related.

In this document, we use the term cable application to include all interactive applications that are deployed on a cable television network. This includes software developed according to the OCAP™ specification [OCAP] (branded for the viewer as “<tru2way™>” applications). It also includes applications developed according to the ETV-BIF specification [ETV-BIF] and any other language specifications that are generally supported by cable platforms in the future.

The term “cable platform,” as used in this document, refers to the viewer’s system on which these applications execute. This platform is nominally a set-top box connected to a television, but other cable platforms may include a digital TV, a game console, or home computer, connected to the cable network.

This document is intended to provide common guidelines for all interactive cable applications, regardless of the platform on which they execute. In addition, this document will provide platform-specific details for authors of ETV and OCAP applications where appropriate and relevant.

The audience for this document is application developers and engineers.

1.2 Purpose of document

The purpose of this document is to provide detailed guidelines to aid software developers in the creation of interoperable, well-behaved cable applications. This document describes how to achieve the results recommended in the [UEG] document, and otherwise interoperate with other cable applications.

2 REFERENCES

2.1 Informative References

This guideline document uses the following informative references.

[CANH-API]	Conditional Access Handler API Specification, CLP-SP-CANH-API-I01-081217, December 17, 2008, Cable Television Laboratories, Inc.
[DVB-MHP 1.0.3]	DVB Multimedia Home Platform 1.0.3, DVB-MHP 1.0.3, ETSI TS 101 812 V1.3.2 (2006-08)
[ETV-BIF]	Enhanced TV Binary Interchange Format 1.0, OC-SP-ETV-BIF1.0-I05-091125, November 25, 2009, Cable Television Laboratories, Inc.
[ETV-AM]	Enhanced TV Application Messaging Protocol 1.0, OC-SP-ETV-AM1.0-I05-091125, November 25, 2009, Cable Television Laboratories, Inc.
[OCAP]	OpenCable Application Platform Specification, OCAP 1.1 Profile, OC-SP-OCAP1.1.2-090930, September 30, 2009, Cable Television Laboratories, Inc.
[UEG]	Cable Application User Experience Guidelines, OC-GL-UEG-V01-090619, June 19, 2009, Cable Television Laboratories, Inc.

2.2 Reference Acquisition

- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027; Phone +1-303-661-9100; Fax +1-303-661-9199; <http://www.cablelabs.com>
- ETSI, www.etsi.org

3 TERMS AND DEFINITIONS

This specification uses the following terms:

Bound Application	A bound application is bound to, or associated with, the currently tuned channel. When the viewer tunes away from the current channel, the bound application goes out of scope and SHALL be terminated unless it is also associated with the newly tuned channel.
Content	Content is typically used to refer to audio, video, and graphic materials used by a service. Sometimes data and applications are also grouped into this term.
Enhanced TV (ETV)	A general term that refers to interactive services and applications that are provided in conjunction with video programming.
Enhancement	A software application that executes in conjunction with video programming.
Focus	The application with Focus is the one drawn on the top of the stack, which will receive event notifications for all remote control keys in the Focus Key group.
Focus Key Group	The set of remote control key events that will be sent to cable applications when in focus.
graphics plane	The visual layer in the cable platform in which all application elements are rendered. It is displayed on top of the video plane.
Navigator	A navigator is an unbound application, provided by the network operator, that the viewer can activate at any time. The navigator is used to select services and launch other applications.
Prompt	An application element that signals to a viewer that an application is present and may be launched.
Select	An action by the viewer that indicates a particular choice. For instance, a viewer may press a remote control key to choose an action represented by the currently highlighted button in an application.
Select Key	A key on the remote control that is used to select a choice or indicate an action to be performed by an application. NOTE that the Select Key is a logical definition and that the actual key on the remote control may be labeled as “Select,” “OK,” or “Enter,” depending on the manufacturer and model of the device.
Unbound application	An unbound application is not associated with the currently selected channel and does not go out of scope when the viewer tunes away from any specific service.
User Interface	A presentation generated by an application that includes visual elements and responds to viewer actions.
Video Plane	The visual layer in the cable platform in which all broadcast video is played. It is displayed below the graphics plane.
Visual Element	A graphical object generated by an application. Examples include a text string, a shaded rectangle, and a bitmap.
Window	The area in the graphics plane in which an application renders all of its user interface elements.

4 ABBREVIATIONS AND ACRONYMS

This guideline document uses the following abbreviations:

AWT	Abstract Windowing Toolkit
DVR	Digital Video Recorder
ETV	Enhanced TV, see above
GUI	Graphical User Interface (see User Interface, above)
MSO	Multiple Service Operator, same as cable network operator
OCAP	OpenCable Applications Platform
PNG	Portable Network Graphics
UA	User Agent
UI	User Interface, see above
WBA	Well-Behaved Applications

5 CONVENTIONS

The following conventions are used in this document:

5.1 Specification Language

The following words are used throughout this document to define the significance of particular requirements:

SHALL	This word means that the item is an absolute requirement of this specification.
SHALL NOT	This phrase means that the item is an absolute prohibition of this specification.
SHOULD	This word means that valid reasons may exist in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
SHOULD NOT	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
MAY	This word means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

6 WBA APPLICATION ENVIRONMENT

Cable platforms are divided into two categories: legacy devices and tru2way (OCAP) devices. Where applicable, the following sections will describe guidelines for applications running on both types of devices. When application portability techniques diverge between legacy and tru2way devices, this document provides device-specific guidelines.

6.1 Legacy Cable Platform Application Environment

Legacy cable devices are, by definition, all set-tops that do not include the OCAP execution environment. These set-tops run a variety of operating systems on a variety of processors and hardware designs. To address the key elements of interoperability, these devices all support the ETV Binary Interchange Format (EBIF) platform for executing cable applications. This section will address specific elements of interoperability in this environment.

6.1.1 Legacy Cable Platforms

There are dozens of models of set-tops in the market that do not run OCAP, but which are suitable for running EBIF applications. EBIF is considered to be the lowest common denominator for a common application platform across ALL cable devices.

Because cable devices have a very wide range of capabilities, applications must use a number of techniques for customizing their behavior. The first step is, of course, identifying the platform on which your application is running. To make the process somewhat easier, [ETV-BIF] specifies three platform profiles and designates the category or profile for each platform:

- **Baseline Profile** platforms include both the Motorola DCT2000 and Scientific-Atlanta Explorer 2000 series devices, as well as a handful of other models. Key features are low memory and reduced color palettes.
- **Full Profile** platforms include a wide range of middle-tier devices. These devices tend to have more available DRAM (dynamic random access memory) and can reproduce thousands (if not millions) of colors in the graphics plane.
- **Advanced Profile** platforms have essentially tru2way-equivalent feature sets. Memory is not typically a constraint, and drawing uses a 24-bit Truecolor palette.

For a complete table of all platform capabilities and their implementation status within each profile, refer to [ETV-BIF] Annex H – Platform Parameter Sets and Annex I - Minimum Profile Requirements.

6.1.2 Memory and Application Size Constraints

One of the major limiting factors in developing applications for the very low end (Baseline) platforms is the lack of DRAM on those devices. This acts to limit the size of applications that target baseline platforms. [ETV-BIF] specifies that the lowest end devices are only required to have a 64 KB cache for holding application resources. This includes all resource, font, and image files which are to be resident in memory at a given time. Full and Advanced profile devices will have at least a 256 KB Resource Cache for holding applications and their data.

When writing EBIF applications, there are at least two techniques for accommodating the lower memory budget on the lowest end baseline devices. One technique involves coding the application slightly differently for baseline devices versus the more robust target platforms. All “versions” of the application are broadcast at the same time, but the EBIF UA running on each platform is able to selectively load the proper subset of resources that are targeted towards that device. Using this technique, an application could use, for example, richer background images on higher end boxes, but use solid color backgrounds on the lower end boxes, in order to save memory.

The second technique is to divide an application into multiple page resources that would be loaded and unloaded dynamically, during the application lifecycle. An application could load the primary page resource, but then have that page load one of several other pages, rather than trying to fit more complex logic onto a single page.

6.1.3 Concurrent Application Execution

Another aspect of the limited memory on baseline devices is that only one application will ever be active and/or visible at a time. If there are two EBIF applications signaled within a Service, or there are unbound applications running when a bound application is signaled, it may be necessary to completely unload one of those applications in order for the second one to run. This can cause extra delays when swapping back and forth between these or other applications.

Annex I of [ETV-BIF] lists the Minimum Platform Requirements for each profile. A Baseline Profile device is not required to even have two applications loaded at the same time, and so will make implementation-dependent decisions about which application to execute when there is more than one choice.

A Full Profile device is required to be able to have multiple applications resident in memory, though only one is required to be active. When a new application is activated, any running applications are likely to be suspended.

6.1.4 Color Space Considerations

The second consideration for ETV applications is the use of colors for drawing. The baseline platform allows only 16 colors for drawing, so fewer options are available to the application developer.

6.1.5 Resolution Considerations

Legacy Motorola and Scientific-Atlanta set-tops have different graphics resolutions. The Motorola systems display graphics at 704x480 (or 352x240) pixels. The S-A systems display graphics at 640x480 (or 320x240) pixels. Developers building interoperable applications must be sensitive to the actual screen size when laying out components, with the realization that the application must support multiple configurations.

6.1.6 Remote Control Key Event Considerations

All cable platforms have one standardized user input device; a remote control. Applications receive events from the platform when the buttons on the remote control are pressed and released. Specific techniques to be used for receiving and processing key events will be discussed later in this chapter.

There is no particular standard for the number of buttons (or keys) or the labels on them among remote controls used with legacy set-tops. This creates problems for application developers who wish to prompt the user to press a specific key that may or may not have a specific label. There is also no guarantee that applications will be able to receive events for all keys on the remote. Service operator specific applications will often reserve several keys for their private usage. For a list of those key events which are generally available to all applications, see [UEG].

6.1.7 Image Processing

The ETV application environment includes support for decoding and displaying PNG images. There are, however, some restrictions on how different types of PNGs are processed on different devices. There are three types of PNG images that are discussed in this context:

- Color Type 2 (Truecolor)
- Color Type 3 (Indexed Color)
- Color Type 6 (Truecolor with Alpha)

The following chart indicates support for each of these types on each ETV device profile.

Table 1 - PNG Support in ETV

	Color Type 2 (Truecolor)	Color Type 3 (Indexed-color)	Color Type 6 (Truecolor with alpha)
Baseline	no	yes	no
Full	yes	yes	yes
Advanced	yes	yes	yes

Baseline profile devices are only required to support PNG color type 3 images, while Full and Advanced profile devices are required to support PNG color type 2, 3, and 6 images.

6.1.7.1 Baseline Profile

Baseline profile devices are only required to support PNG color type 3 images. These are *palletized* images, and typical User Agent behavior is to map the indices of the PNG color palette to the application-specified palette. For best results, the PNG image should be authored using the same palette (same colors in the same order) as the application. This will yield the most predictable result, since the colors in the PNG will be mapped to the correct colors in the application palette. If the PNG image is authored with a palette that is different than the application palette, the PNG image will be displayed using different colors than intended due to the color mapping mechanism.

6.1.7.2 Full Profile

While Full profile devices are required to support PNG color types 2, 3, and 6, User Agents running on this middle range of devices typically exhibit the greatest variation in display behavior.

PNG color type 2 images may specify a single RGB value in the transparency chunk, which will be interpreted as fully transparent to graphics.

PNG color types 3 and 6 can contain additional transparency information, either in a transparency chunk (color type 3) or on a per-pixel basis (color type 6). In this context, an alpha value of 0 will be treated as fully transparent to graphics, an alpha value of 255 will be treated as fully opaque, and an alpha value between 1 and 254 will be treated as translucent to video.

Indexed images (PNG color type 3) may preserve the image palette on some Full or Advanced User Agents, rather than map to the application palette. This is another reason to make sure that the image palette matches the application palette: otherwise, the image may appear differently on different devices.

To support transparency for PNG color type 3 images, some User Agents further require that the application palette have an entry (of a transparent to graphics color) of the same index as the transparent color in the image palette, and also a transparency chunk specifying an alpha value of 0 for that color.

Due to hardware limitations, some platforms may only support a single level of translucency to video at any given time. On these platforms, the first application palette entry that specifies partial translucency to video will determine the alpha level for any subsequent partial translucency to video palette entries. In order to achieve consistent application behavior and display, the *application authored PNG translucency to video* metadata item (see below) can be used to override this mechanism and force a single partial translucency to video.

6.1.7.3 Advance Profile

Advanced profile devices are also required to support PNG color type 2, 3, and 6 images. These platforms are typically the most capable in terms of hardware capabilities, and these devices tend to exhibit consistent rendering behavior. To achieve the greatest consistency, image resources targeted to Advanced profile devices should use PNG color type 6. Because each pixel of these images contains RGB and alpha information, User Agents (and the underlying platforms) can render these images as designed, and no color mapping is necessary.

6.1.7.4 Additional Considerations

6.1.7.4.1 Application Authored PNG Translucency to Video Metadata Item

If the *application authored PNG translucency to video* metadata item is specified, Full profile devices will display any and all translucent pixels (in PNG color type 3 and 6 images) at the translucency value specified. Note that since valid values for this metadata item range from 0 - 127, the specified value will be linearly scaled to a 0 - 255 value range. For example, 0 specifies fully transparent, 127 specifies fully opaque, and 63 specifies 50% translucency to video. Also note that when running on a Full profile device, the value of this metadata item is also applied to any graphics pixel of an EBIF widget (border, background, text, etc.) which uses a palette entry containing partial translucency to video (1 through 126).

This metadata item is disregarded when the application is running on a Baseline or Advanced profile device.

6.1.7.4.2 Palette Entry 0

Application palette entry 0 is reserved for fully transparent to video. The impact of this is that any pixel in a PNG image that shares the same RGB (red, blue, green) value as palette entry 0 will not be rendered, and the effect is that video will be displayed through the image for those pixels. Some toolkits restrict the RGB value of palette entry 0 to (0, 0, 0) - pure black. If authoring in such a toolkit, care must be taken when creating PNG images. Unless the intended behavior is to "punch through" to video, pure black (0, 0, 0) should be avoided when creating PNG images. If the application authoring toolkit provides the ability to modify the RGB value associated with palette entry 0, this palette entry can be modified to use a color that is not present in any PNG images. A common and recommended practice is to use "hot pink" (255, 0, 255) as the RGB value for palette entry 0, since this color is typically not present in images.

The transparency to video mechanism works for all three types of PNG images (color types 2, 3, and 6).

6.2 OCAP Application Environment

All newer set-tops designed for U.S. cable include the OCAP middleware. OCAP is a standardized application platform defined by [OCAP]. Tru2way is a brand name for products that are compliant to [OCAP] and other OpenCable specifications.

6.2.1 Service Provider Applications

One means of categorizing applications in the OCAP environment is in terms of their privileges. An application with more privileges has greater access to device resources. Privileges are granted to applications at the time that they are published, either directly by the MSOs, or indirectly, by CableLabs. In order for an application to be granted even very general privileges, it must demonstrate to one or more MSOs that it is "well-behaved".

Those applications with the most privileges are those that are published by the service operator directly. These Service Provider Applications (SPAs) include the Monitor Application and Navigator, as well as one or more Presentation Engines, such as the EBIF UA. See Figure 1.

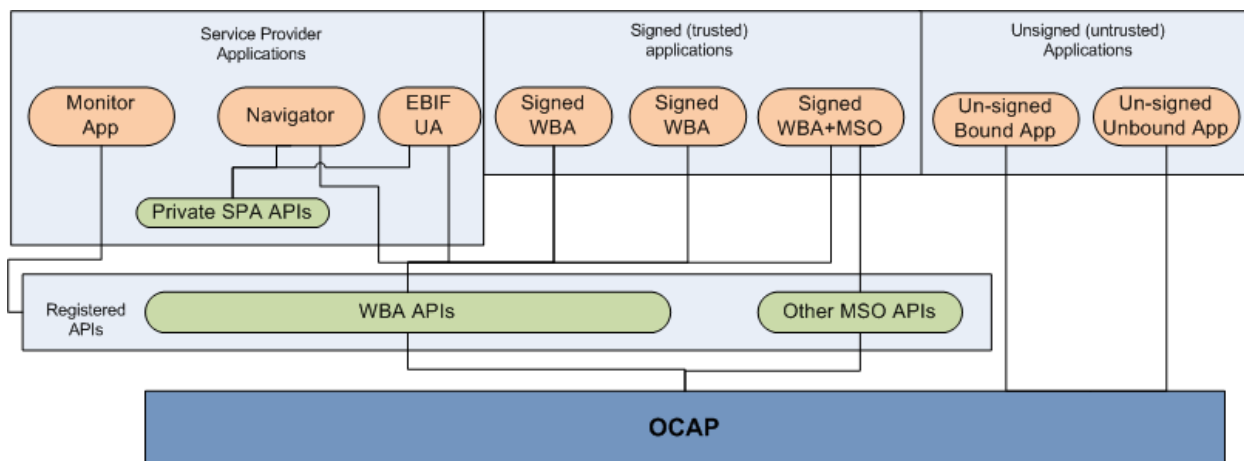


Figure 1 - Service Provider Applications and Registered APIs

6.2.2 Service Provider APIs

Besides performing their primary duties, the SPAs also provide services to other applications that are running in the system. The SPAs will install several shared libraries (registered APIs) for accessing various types of MSO-specific data or for providing inter-process communication services on behalf of one or more calling applications.

- **WBA APIs** - The SPAs play a critical role by publishing a number of APIs that other applications can use in order to be more “well-behaved”. These APIs implement a set of functionality that was left out of early versions of OCAP, but which greatly enhance the ability of applications to co-exist. Through these APIs, the SPAs are able to exert a certain amount of control over other applications, or operate on their behalf, thus helping to enforce the well-behaved nature of third-party applications not created by the MSO.
- **Other MSO APIs** - The SPAs may publish other APIs related to Guide data services, parental control ratings, persistent storage, etc. Such APIs fall outside of the scope of the WBA Guidelines.

Access to each set of these APIs is controlled by specific privileges. In order for an application to be granted access to the WBA APIs, it must be signed. Un-signed applications will not have access to those special APIs nor many other system resources.

The WBA APIs expose new functionality to applications related to focus management, key event processing, and access to the video subsystem.

6.2.2.1 WBA API Access Permissions

In order to gain access to the WBA APIs, an application SHALL have the following permission in its Permission Request File (PRF):

```
<ocap:registeredapi.user name="com.opencable.wba"/>
```

6.2.2.2 Focus Management and Key Event Processing

This part of the WBA APIs gives an application finer grain control over AWT Focus and Z-ordering of application windows, when more than one application is present and visible.

The WBA APIs also provide an enhanced Key Event processing facility, which can conditionally make some “MSO Exclusive” key press events available to other applications when they are in focus and the MSO applications are not otherwise using those keys.

For details on Focus and Z-Order Management, see Section 6.4, Window and Focus Management Guidelines.

6.2.2.3 Centralized Video Service Management

With the proliferation of content across local storage devices, on-demand libraries, switched broadcasts and home media servers, it is increasingly complex for an application to develop all of the code necessary to select and control

the playback of that video. Moreover, applications that have unfettered control over the video plane can often leave the system in an unpredictable state when transitioning unexpectedly to the background. As a means of providing a more consistent user interface, as well as simplifying the development of most applications, the WBA APIs also introduce a common service known as the Shared Video Manager (SVM).

The SVM allows third-party applications to select broadcast services, recorded services, and on-demand services, and to control the rate of playback, the video scaling, audio language, etc, using a vastly simplified interface. If there are parental control challenges, CA issues, or tuner conflicts, those are all dealt with by the SVM, which then signals the application of the ultimate outcome. The use of this facility greatly simplifies the issues of consistency around user interfaces with regard to service selection.

For details on the use of the SVM, see Section 6.7.3, OCAP Shared Video Functionality.

6.2.2.4 Inner Applications

With the growing number of EBIF applications and developers, there is a growing interest in embedding EBIF content within an OCAP/Java application. This technique extends from an MHP facility known as Plugins and InnerApplications. In this case, the SPAs install a version of the EBIF User Agent that is configured as a Plugin, making it available to other Well-behaved Applications to render EBIF “Inner Applications” from within their OCAP application.

Optionally, an MSO may package additional Plugins that may interpret any number of other byte-code or script based formats. Because the User Agent may be provided by a number of different vendors, these guidelines address the basic properties of interoperability with those User Agents packaged as Plugins.

To use this facility, an application SHALL have the following permission in its Permission Request File (PRF):

```
<ocap:registeredapi.user name="com.opencable.wba.plugin"/>
```

6.2.2.5 CANH Libraries

Another API installed by the SPAs is the CA Network Handler (CANH) Library. This library gives applications access to status events from the CableCARD™ concerning authorizations to view subscription services, access Pay-Per-View channels, or to gain access to specific application features or capabilities.

The design of the system allows only one CAHandler object to be created, but an SPA (typically the Monitor Application) will instantiate the CAHandler and register it with the IXC Registry, so that other applications may have access to the CAHandler through a remote interface.

For details on the use of the CANH Libraries, see [CANH-API].

To gain access to the CANH libraries, an application SHALL have the following permission in its Permission Request File (PRF):

```
<ocap:registeredapi.user name="com.opencable.canh"/>
```

6.3 Application Lifecycle Guidelines

6.3.1 OCAP Application Lifecycle

An OCAP application has four states: loaded, active, paused, and destroyed. See [DVB-MHP 1.0.3] clause 9.2.3.2 for a description of these states, related transitions, and entry points associated with each transition.

The base class for all OCAP applications is javax.tv.xlet.Xlet. This section discusses the Xlet execution states and recommended or required behaviors for a well-behaved application during each state transition.

The Xlet interface defines five entry points:

- A no-argument constructor
- `initXlet()`
- `startXlet()`
- `pauseXlet()`

- `destroyXlet()`

These methods are called by the OCAP implementation based on various events and conditions, as described within each section.

Applications SHALL NOT intentionally block, including invoking `Thread.sleep`, during any of these methods, as this may delay the OCAP implementation from starting other processes or applications.

6.3.1.1 **Constructor**

When the OCAP implementation first loads an application to prepare it for execution, it calls the no-argument constructor for the base class of the application. This constructor is not expected to do anything, and need not be implemented by the application. Instead, all application initialization is expected to be handled by the `initXlet()` method.

The one exception to this convention is the Initial Monitor Application, which SHOULD implement the no-argument constructor, during which it SHOULD call `OcapSystem.monitorConfiguredSignal()`.

6.3.1.2 **initXlet**

`initXlet()` is the initial entry point for an OCAP Application. It will only be called once, so an application need not check for re-entrance.

Upon the successful return of `initXlet()`, the application is in the *paused* state. Typically an application is expected to release resources and remain quiescent while in a *paused* state; see [OCAP], sections 9.2.3.2, 9.3.3.3.2, and A.5.4.2.

However, these rules do not apply during `initXlet()`, as this method is followed by an immediate call to `startXlet()`. Applications are allowed to prepare for an imminent transition to the *active* state. Typical tasks during `initXlet()` may include:

- Initialization of application variables.
- Requesting access to the Root Container for the application and setting its bounds.
- Adding the Xlet Component to the Root Container (but not making the Root Container visible).
- Creating DVB UserEventRepositories for all key groups that will be managed by the application (but not requesting exclusive access to those events or requesting AWT focus).
- Loading files necessary for the application to run. It is recommended to use background threads for any extended I/O operations at this point, returning from `initXlet()` as quickly as possible.

While it is possible for applications to receive AWT focus events or user input events while in the paused state, an application SHALL NOT request focus, install AWT or DVBEvent listeners, or make itself visible within `initXlet()`.

6.3.1.3 **startXlet**

The `startXlet()` method is called in order to transition the application to the *active* state. The application transitions to this new state when `startXlet()` returns without an Exception.

An application SHOULD perform the following operations within `startXlet()`:

- Allocate all major blocks of memory.
- Start any background threads for loading data from the file system.
- Install all event listeners.
- Make the initial GUI components visible.
- Request AWT focus, if using AWT events.
- Otherwise prepare the application for execution.

`startXlet()` may be called more than once in an application's lifecycle. The [DVB-MHP 1.0.3] specification indicates that `startXlet()` will be called immediately after the successful return of `initXlet()` and also when returning from a *paused* state.

There are two reasons that an application may be transitioned to an *active* state. The first is that the application's home environment has become selected. The second reason is in response to a programmatic request to become active. An application may request its own transition to an active state from a paused state by using `XletContext.resumeRequest()`. An application may be activated by a second application using `appProxy.start()` and `appProxy.resume()`.

It is not guaranteed that the resume requests will be honored by the OCAP implementation, but this request typically results in an immediate call to `startXlet()`.

An application **SHOULD** take care to only re-initialize objects or data that were released while in the *paused* state. A simple solution is to keep a boolean flag that indicates whether `startXlet()` has already been called.

An application **SHALL** be prepared for multiple calls to `startXlet()`. An application **SHALL** reinitialize any and all components disabled by an intervening call to `pauseXlet()`.

6.3.1.4 *pauseXlet*

The `pauseXlet()` method is called in order to transition the application to the *paused* state. The application transitions to the new state when `pauseXlet()` returns.

There are two reasons that an application may be transitioned to a *paused* state. The first is that the selected cable environment no longer supports the application's home environment. The second reason is in response to a programmatic request to pause. An application may request its own transition to a paused state from an active state by using `XletContext.notifyPaused()`. An application may be paused by a second application using `appProxy.pause()`.

The implementation may unilaterally remove resources from an application in the paused state. See [OCAP] sections 11.2.2.3.19, 10.2.2.4, and Annex Y.

While applications are expected to minimize their resource use, they should keep themselves ready to become active again. These contradictory imperatives lead to a non-deterministic recommendation to give up any buffers or data elements that can be restored quickly. Those that require more lengthy operations or access to remote files should be maintained, if at all possible. As an example of this, consider an application that keeps a large "raw" data buffer and periodically uses that data to create a series of objects, based on the application context and state. Keeping the raw data is probably more important than the cache of objects that were contextually generated based on that data.

An application **SHOULD** release all reserved resources during `pauseXlet()`.

An application **SHOULD** be prepared to lose any reserved resources while in the paused state that were not released during `pauseXlet()`.

An application **SHALL** hide its UI and **SHALL NOT** request AWT focus while paused.

6.3.1.5 *destroyXlet*

This entry point is called when the application is to be destroyed. `destroyXlet()` is called with an *unconditional* parameter that indicates whether this is a suggestion or a demand. If unconditional is set to true, the implementation will terminate the application. If unconditional is set to false, the application can delay its termination by returning with an `XletStateChangeException`.

An application **SHALL** follow the directions for this state transition as described in [DVB-MHP 1.0.3] clause 11.7.1.2:

- Cause any threads that it has created to exit voluntarily.
- Stop, deallocate and close any JMF players that it has created.
- Stop and destroy any JavaTV service selection `ServiceContext` objects it has created.
- Release any other scarce resources e.g., `NetworkInterfaceControllers`, that it has reserved.
- Flush any images using the `Image.flush()` method.
- De-register any event listeners.

- Terminate any active timers.

In addition to these tasks, an application SHALL NOT sleep, or more generally, SHALL NOT cause any unnecessary delay during `destroyXlet()`.

An application MAY save any state information in persistent storage in the event that it is re-started in the future and wishes to resume from where it left off.

An application can terminate itself by calling `XletContext.notifyDestroyed()`. In this case, the implementation will NOT call the `destroyXlet()` entry point, so the application SHOULD perform all of these functions before calling `notifyDestroyed()`.

6.3.2 Bound Application Prompts

Refer to [UEG] for general guidelines on the display of prompts or teasers for bound applications.

6.4 Window and Focus Management Guidelines

6.4.1 Window and Focus Management Model

One of the key tenets of application interoperability is defining when and how applications are allowed to interrupt or visually occlude each other. The visual model of cable application platforms is described in [UEG]. Essentially, those guidelines describe a system of overlapping windows.

- Each application exists within one window, though it can have many overlapping visual elements within that window.
- An application can ask for its window to be put at the front of the stack or at the back of the stack.
- Applications at the top of the stack will have Focus for receiving AWT KeyEvents.
- Applications are drawn from the back to the front.
- Although it is not technically mandated, it is recommended that Bound applications display themselves initially to the back of the stack, so as to have the least visual priority compared to other active applications.

6.4.2 Window and Focus Management on OCAP Platforms

The WBA APIs were designed, in part, to address the lack of a Window Manager in OCAP. Although the intent is for this functionality to be added into the OCAP specification in the future, the WBA APIs provide a workable set of methods for cooperative behavior in the OCAP environment.

The WBA APIs introduce the `OCSceneManager`. The `OCSceneManager` acts to control focus and Z-Order among all well-behaved cable applications. The `OCSceneManager` is not a part of the public API, but is instantiated by an MSO application (typically the Monitor App) when it initializes the WBA library. `OCSceneManager` uses common inter-process communication techniques (IXC or sockets) for receiving requests and sending events to each application that requests its services.

6.4.2.1 OCScene and Focus Management

In OCAP, the root Container for all application graphics is an `org.havi.ui.HScene`. In order for this functionality to be added to the OCAP stack, `HScene` must be modified. To accommodate this need, the WBA libraries define the `org.cablelabs.oc.wba.ui.OCScene` class, which acts as a proxy for the `HScene`.

`OCScene` will instantiate the default `HScene` on behalf of simple applications, or allow more sophisticated applications to create and specify an `HScene` based on alternative configurations. When an application launches, instead of asking OCAP for its `RootContainer` (an `HScene`), it SHOULD instantiate an `OCScene` (which in turn will associate itself with the `RootContainer`). If using the default `HScene` is not sufficient, an application can create an `HScene` and pass it to the `OCScene` for its use.

Internally, the `OCScene` object communicates with the `OCSceneManager` to maintain the stack of all Well-Behaved Applications (WBAs).

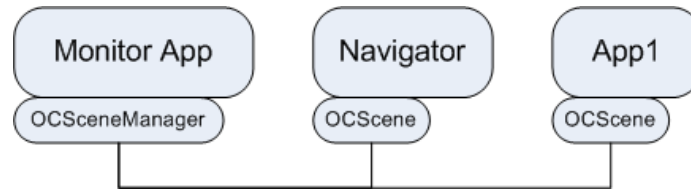


Figure 2 - OCSScene

OCSManager is primarily intended to address an ambiguity in AWT and OCAP concerning visibility and focus. It is possible, in OCAP, for an application to have focus (and to therefore receive the Focus Key input events) even though it is not the front-most application on the stack. OCSManager assures that the front-most application on the stack will always have focus. OCSManager manages this by always addressing focus and visibility in tandem. Rather than leaving control of focus to the application's discretion, OCSManager specifically requests focus for the front-most application on the stack.

As a side effect of this, the application should never request focus on any of its other components; to do so would bypass WBA's focus management system. The application can, however, register for FocusEvents by adding a FocusListener to the HScene. FocusEvents are used when there are multiple applications running, as a sure indication that an application is visible, on the top of the stack, able to use active rendering, and able to receive all Focus Key Events and potentially to be able to receive some MSO Key Events.

6.4.2.1.1 ShowToFront

To address the ambiguity in the definition of show() and setVisible(true) in AWT and HScene, OCSManager implements the method showToFront(). showToFront() specifically places the OCSManager in the front-most stack position, sets visibility to true, and requests focus for the HScene associated with that OCSManager.

6.4.2.1.2 ShowToBack

To address a lack of functionality in AWT and HScene, OCSManager implements the method showToBack(). showToBack() specifically places the OCSManager in the rear-most stack position, but may not set the visibility flag to true and will not request focus until the application moves to the front of the stack.

If there are no other applications on the stack, or if all other applications are yielding, this method behaves just like showToFront().

6.4.2.1.3 Yield

The yield() method indicates to the OCSManager that this application is willing to yield focus. This call is modal, and places the application in a WILLING_TO_YIELD state. The WILLING_TO_YIELD state can be canceled by a call to showToBack() or showToFront().

If this application is the application in focus and any other applications are on the stack but not in the WILLING_TO_YIELD state, the focus will change immediately. Otherwise, focus will stay with this OCSManager, but its state will be set to WILLING_TO_YIELD. The application can determine whether or not it lost focus, based on the receipt of FocusEvents.

When an application yields and another application is granted focus as a consequence, the yielding application will receive a FocusEvent from the OCSManager indicating that it has lost focus. If no other application was granted focus, the caller will not receive a FocusEvent, and will continue to receive KeyEvents for the focus keys and all registered MSO keys.

In the case of a focus change, this application will move back one spot in the stack. It is not required to be invisible while in the WILLING_TO_YIELD state.

6.4.2.1.4 Hide

The hide() method indicates to the OCSManager that the OCSManager should be removed from the list of visible windows. If the OCSManager has focus, the application will receive a FocusEvent indicating that it has lost focus.

6.4.2.1.5 Top-Of-Stack Resource Contention Management

The `showToFront()` and `showToBack()` methods take a `ResourceClient` parameter. If these parameters are null, no resource contention management is carried out, and the `OCSceneManager` honors all requests equally. If, however, a `ResourceClient` is specified, then normal DAVIC resource contention management operations are carried out. The application that currently has focus can deny another application's request to `showToFront()` until a critical user input operation (for example) is completed.

To protect the interests of the MSO applications, it is likely that the `OCSceneManager` will give preferential treatment to the core MSO applications in resource contention dialogs.

6.4.2.2 OCSceneController

Along with `OCScene`, the WBA Library introduces the `OCSceneController` class for managing visibility and focus. The `OCSceneController` is retrieved from the `OCSceneManager` by an MSO application (typically the Navigator) that acts as a master control application. `OCSceneManager` allows one and only one application to ask for the `OCSceneController` and will deny subsequent requests from other applications.

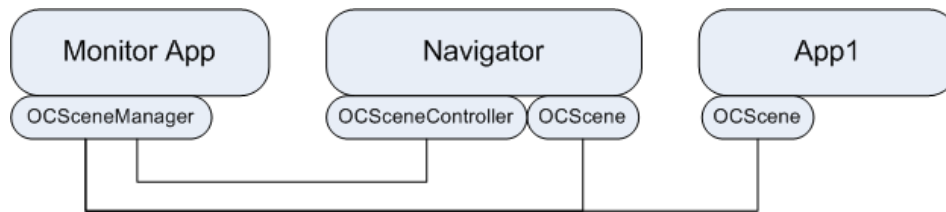


Figure 3 - OCSceneController

Because the Navigator is typically the application that is launched first, it should be able to guarantee access to the `OCSceneController`. All other WBAs SHOULD NOT attempt to gain access to the `OCSceneController`.

6.4.2.2.1 getAppList

`OCSceneController` supports a method called `getAppList()` which returns a list of all WBAs, represented as an array of `OCSceneBindings`. From the `OCSceneBinding`, the MSO application can determine the position on the stack, visibility, yield mode, and the `orgID/appID` of each active application.

6.4.2.2.2 ShowToFront

After retrieving the list of all applications, it is possible for the `OCSceneController` to move a specific application to the top of the stack, thereby directing input focus to that application. This call works the same as if the target application called `showToFront()` on its own `OCScene`.

6.4.3 User Input Model

6.4.3.1 Focus Keys

Some key events will be reserved by MSO applications (or consumed by the device itself) and will not be generally available to other applications. A subset of all user input keys has been identified which will be available to all cable applications when they are in focus. For a description of Focus, see [UEG] or Section 6.4 Window and Focus Management Guidelines of this document.

The Focus Keys are:

- Up Arrow
- Down Arrow
- Left Arrow
- Right Arrow
- Select
- Info

- Exit
- Function Key 0 (Red)
- Function Key 1 (Green)
- Function Key 2 (Blue)
- Function Key 3 (Yellow)
- Page Up
- Page Down

Note: Some legacy cable platforms have only three function keys (Red, Blue, and Yellow). All tru2way devices will have four function keys on the remote control (Red, Green, Blue, and Yellow).

6.4.3.2 Other Keys

As a general rule, all other remote control input events will be reserved by the MSO applications running on the device, although most of those keys will be shared, if requested, when the application is in focus. Of the non-focus keys, MSO applications are expected to permanently and unconditionally reserve the following key events:

- Guide
- Menu
- Channel Up
- Channel Down
- Volume Up
- Volume Down
- Power

The other non-focus keys are delivered to a third party application only at the discretion of the MSO applications that are reserving those keys. Examples of these keys are the numeric keys (0-9) or the VCR Shuttle keys (Play/Pause/Stop/Record/FF/Rewind). These key events are shared at the discretion of the MSO applications.

6.4.3.3 User Input Model for OCAP

OCAP has two facilities for receiving key press events: AWT KeyListener and the org.dvb.event.EventManager. Generally, the EventManager interface should be reserved for MSO applications. Third party applications SHOULD NOT reserve key press events through the EventManager. If an application does use the EventManager, it SHALL only reserve the Focus Key events while it is in focus. When an application detects that it no longer has focus, it SHALL release any reservations on the Focus Key events.

6.4.3.3.1 MSO Key Forwarding Using the WBA APIs

As an alternative to the DVB EventManager, MSO key events may be requested by applications through the WBA libraries. Again, this facility is implemented via OCScene and OCSceneController.

6.4.3.3.2 OCScene.setKeyListener()

When an application is interested in receiving MSO key events, it calls OCScene.setKeyListener. The calling application specifies the list of KeyEvent in which it is interested and a KeyListener interface for receiving those KeyEvents.

6.4.3.3.3 OCSceneController.processKeyEvent

When the MSO application that has the OCSceneController is not in focus, and is not otherwise interested in consuming a specific KeyEvent, it can, at its discretion, pass that KeyEvent on to other WBAs. This method turns the KeyEvent over to the OCSceneManager, which will direct that event to the application on the top of the stack.

If the application on the top of stack has registered with OCScene.setKeyListener(), its keyTyped() method will be called with this KeyEvent. If the application uses the KeyEvent, it should mark it as “consumed” before returning.

If that application is not interested in the given KeyEvent (i.e., does not mark it as consumed), it will be returned to the MSO Application for its own use.

As an example, the Navigator has registered as the OCSceneController. The user is watching TV when a bound application appears and asks the user to press a numeric key. If that bound application has registered interest in the numeric keys (through OCScene.setKeyListener), the KeyEvent will be passed to that application. If the application

uses (consumes) the key event, the Navigator SHALL NOT use that key event itself. If the application is not interested in the key, the navigator may use that key press as an indication of a channel change request.

6.4.3.4 User Input Model for EBIF

In EBIF, there is a specific Action Event called OnKeyEvent, which, as the name implies, is used for triggering event processing subroutines. While well-behaved applications are guaranteed to be able to get the Focus Key events, it is also likely that an EBIF application will receive the numeric keys (0-9) and the VCR Shuttle control keys (play, pause, etc). This implies that the ETV User Agent running on OCAP devices has registered an appropriate KeyListener through its OCScene. For legacy devices, this implies that the Navigator is not otherwise permanently reserving and consuming those associated KeyEvents.

6.5 Graphics and Drawing Guidelines

6.5.1 HScene Clipping

OCAP defines clear mechanisms for an application to control alpha-blending between the graphics that it draws into its own HScene, as well as between the graphics plane (OSD) and the video plane. Furthermore, OCAP gives applications the ability to control blending between each other, when more than one application is visible on the screen at a time.

Inter-HScene blending rules are consistent with intra-HScene blending rules. The display is created from a composite of all visible HScenes. When each application renders its HScene, it can define each pixel to be:

- unmodified
- modified
- replaced

The pixel is unmodified when the HScene background mode is set to `BACKGROUND_NO_FILL` *and* the application does not explicitly render that pixel.

The pixel is modified when the HScene background mode is set to `BACKGROUND_NO_FILL` *and* the application renders that pixel in `SRC_OVER` mode.

The pixel is replaced when the HScene background mode is set to `BACKGROUND_FILL` *or* a background image is defined that covers that pixel *or* when the application renders that pixel in `SRC` mode.

The following figures show two applications' HScenes blended together with `SRC_OVER` (first over a white background, then over video).

Note that:

- Not only is the background scene visible through the non-rendered areas of the foreground scene, but it is also visible through the translucent blue pixels of the foreground scene.
- Where the translucent pixels of the foreground application mix with the translucent pixels of the background, video can be seen.
- Where the translucent pixels of the foreground application mix with the opaque pixels of the background, video cannot be seen.

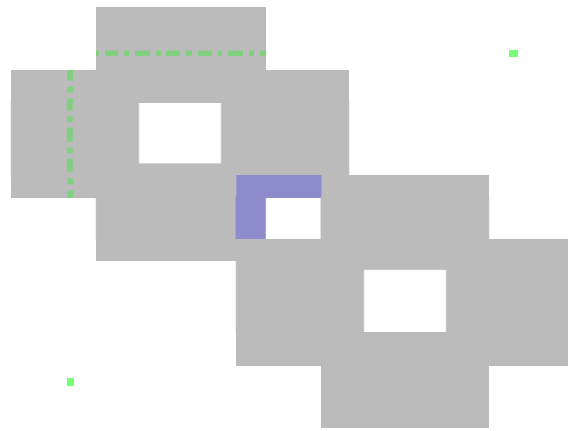


Figure 4 - Fully Blended scenes



Figure 5 - Fully Blended scenes over video

6.5.2 Repairing Overlays

When an application makes its HScene invisible or inactive, the OCAP implementation will recomposite the display by calling the paint() method of the root component of each visible HScene. The Graphics object passed to paint()

may optionally be clipped to the portion of the display that was obscured, but would otherwise be clipped to the size of the HScene.

Alternatively, it is possible that the OCAP implementation will keep frame buffers for each visible HScene, and reconstruct the relevant portion of the display on its own. In this case, `paint()` would not be called, and the application would not otherwise know that its graphics were restored or even needed to be restored.

As above, the OCAP implementation will recomposite the display from all visible HScenes, from bottom to top, clipping on the bounds of the HScene.

6.5.3 Active Rendering

Although most graphics drawing is done when the OCAP implementation calls an application's `Component.paint()` method, applications also have the choice of drawing on a separate animation thread. This technique is referred to as active rendering.

While this technique is essential for animations, several rules apply to those application doing active rendering, to prevent those actions from disrupting AWT's control of the other applications being rendered:

- An application SHALL NOT perform any active rendering unless that application has the front-most HScene on the stack of visible HScenes.
- Even when doing active rendering, an application SHALL paint its entire display (subject to clipping) when `paint()` is invoked.
- When doing active rendering, the application SHALL be responsible for all display updates, including clearing or erasing of previous contents, if necessary.
- Active rendering is not expected to play well with inter-scene blending. An application SHOULD disable inter-scene blending via `HScene.setBackgroundMode()` when doing active rendering.

6.6 Pixel Burn-in

Pixel burn-in is a condition caused by displaying a static image on a television over an extended period of time. Normally, this is not a problem, as video typically changes virtually every pixel quite regularly. With static graphics or menus, however, this is not the case. Computers solve this by using a "screen saver", which draws random graphics or displays random pictures across the display periodically.

In order to avoid pixel burn-in, applications that present a UI SHALL NOT present a static graphic for more than 10 minutes at a time. This includes banners behind tickers or other static information windows. Even though the text in the window is changing, the background of the window may be relatively static. Applications should find ways of modifying the color or intensity of the drawn pixels periodically to avoid damaging the display.

6.7 Shared Video Guidelines

The Shared Video Manager subsystem is intended to centralize all tuning and video presentation functionality into a single application, typically the Navigator. There are three parts of the SVM subsystem; `OCSHaredVideoManager`, `OCSHaredVideoHandler`, and `OCSHaredVideoProxy`.

The `OCSHaredVideoManager` implementation is typically instantiated by an MSO application such as the Monitor Application.

The `OCSHaredVideoHandler` is typically implemented by an MSO application such as the Navigator, as it already has primary responsibilities for managing service selection and video presentation (sizing, scaling, aspect ratio, playback rate, etc).

All other applications that wish to select a service or control the video presentation will request an `OCSHaredVideoProxy` from the `OCSHaredVideoManager`. All requests made to the proxy will be forwarded to the handler, which has the power to allow or disallow any requests as they are received.

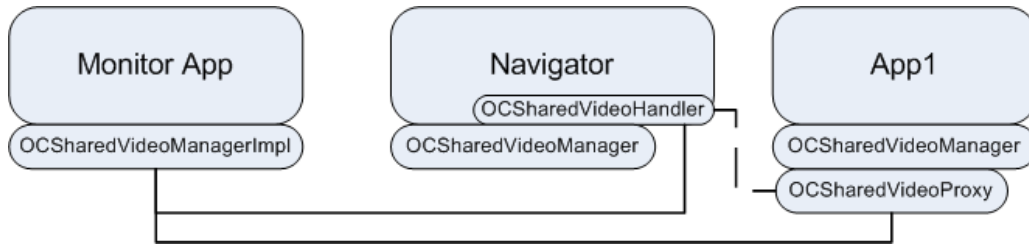


Figure 6 - OCSharedVideoManager

In this case, each application has its own instance of the OCSharedVideoManager in its name space, which communicates with the MonitorApp's implementation of the OCSharedVideoManager. This central implementation handles the registration of remote objects and helps to set up the communications that subsequently occur directly between the OCSharedVideoHandler and all OCSharedVideoProxys.

The WBA library takes care of forwarding the requests from the proxy to the handler, and sending the results from the handler back to the proxy across application boundaries using transport mechanisms defined by the OCSharedVideoManager. All events related to service selection and the video presentation are also forwarded from the handler back to all registered proxies, to enable those applications to follow what is occurring with the video presentation.

6.7.1 Shared Video Proxy

An OCSharedVideoProxy can be obtained by using the following technique:

```
OCSharedVideoManager vManager= OCSharedVideoManager.getInstance(xletContext);
OCSharedVideoProxy proxy = vManager.getSharedVideoProxy(
HScreen.getDefaultHScreen());
```

The getSharedVideoProxy call will pass the specified HScreen and the caller's AppID (obtained from the XletContext) to the shared video handler, which uses the information to decide if it wants to allow that client access to the shared video subsystem.

After retrieving an instance of OCSharedVideoProxy, the client may call methods on that object to manipulate the video display.

6.7.1.1 Registering a Shared Video Listener

Given an OCSharedVideoProxy, applications will typically register a listener for events from the shared video subsystem. A combined listener class called OCSharedVideoListener will handle three of the most common video events:

- ServiceContextListener for ServiceContextEvents
- ControllerListener for ControllerEvents
- VideoFormatListener for VideoFormatEvents

An OCSharedVideoListener must implement the methods from all three listeners above. Once registered with OCSharedVideoProxy.addSharedVideoListener(), it will receive all of these events.

If an application has registered an OCSharedVideoListener, the application SHALL remove the listener before terminating, using the method OCSharedVideoProxy.removeSharedVideoListener().

6.7.1.2 Using Control Proxies

The methods getControlProxy and getControlProxies in the OCSharedVideoProxy class allow an application to get proxies for additional controls to manipulate the video presentation.

There are predefined proxies for the following controls: AudioLanguageControl, AWTVideoSizeControl, BackgroundVideoPresentationControl, ClosedCaptioningControl, DVBMediaSelectControl, FreezeControl, MediaTimeEventControl, TimeShiftControl, VideoFormatControl and VideoPresentationControl.

These controls (and possibly others) may be provided by the handler. A complete list can be retrieved via the method `OCSharedVideoProxy.getControlProxies()` which will return a `Control` array. To get the proxy for a single, specific control, the application would call `OCSharedVideoProxy.getControlProxy(String name)`, where `name` is the fully qualified classname of the control.

A different combination of Controls will be available after each service selection, depending upon the type of service selected. For example, after selection of a Broadcast Service on a device with no DVR capabilities, the Proxy may only return `VideoFormatControl`, `BackgroundVideoPresentationControl`, and `AWTVVideoSizeControl`.

When presenting recorded content, or the contents of a time shift buffer, the Proxy will also return a `TimeLineControl`, `MediaTimePositionControl`, `MediaTimeFactoryControl` and `MediaTimeEventControl`. Other Controls may or may not be exposed, based on the OCAP Stack implementation.

6.7.2 Shared Video Handler

The shared video handler receives requests for video services from various clients and translates them into calls that directly manipulate the video system. The handler can refuse any call at any time, based on its own discretion. After the request to allow a particular application to obtain a Proxy, however, the caller is no longer identified in subsequent calls for service selection or video presentation control.

It is generally expected that the Navigator will allow most requests from proxy applications as long as it is not in focus or displaying an active user interface element.

6.7.2.1 Registration

The shared video handler must be registered with the `OCSharedVideoManager`. Registration is done through a sequence of calls such as this:

```
OCSharedVideoManager svm = OCSharedVideoManager.getInstance(xletContext);
svm.setSharedVideoHandler(HScreen.getDefaultHScreen(), mySVHandler);
```

Note: The `HScreen` parameter to `setSharedVideoHandler` is currently required but ignored. It indicates which screen is being managed. The reference implementation of WBA does not currently support multiple screen devices.

6.7.2.2 Implementation of the Shared Video Handler

In order to implement a shared video handler, a class must be created that implements the `OCSharedVideoHandler` interface and all of its methods.

Most methods provide shortcuts to common services and can be implemented simply: when an application calls a method such as “`selectService(Service)`”, it is forwarded from the `OCSharedVideoProxy` to the `OCSharedVideoHandler`. The handler’s job is to translate that call into code that actually executes the required functionality. The parameters and return value (if any) will be passed to and from the proxy and handler, so most of these functions are very simple to implement; for instance, an implementation of `selectService(Service)` in the `OCSharedVideoHandler` might be as simple as:

```
public void selectService(Service selection) {
    if (inFocus)
    {
        throw new OCRequestRefusedException();
    }
    try {
        mediaPlayer.selectService(selection);
    } catch (Exception e) {
        throw new OCProxiedObjectException(e.toString());
    }
}
```

6.7.2.3 Shared Video Handler Events

The event system is also simplified for the most common types of events. When the application that has an `OCSHaredVideoProxy` calls `addSharedVideoListener`, that listener is passed to the `OCSHaredVideoHandler` via its own `addSharedVideoListener` method.

When the `OCSHaredVideoHandler` next receives one of the Events handled by this listener (`ServiceContextEvents`, `ControllerEvents`, and `VideoFormatEvents`), the handler must start forwarding those events to each listener added by an `OCSHaredVideoProxy`. This is true whether the events were received due to actions initiated by the Navigator itself, or by one of the shared video proxies.

When `removeSharedVideoListener` is called, the listener must be removed from the list of active listeners to prevent further exceptions.

6.7.2.4 Control Proxies

With the majority of the standard Controls, requests for information are channeled from the proxy to the handler. For example, the `VideoFormatControl` has methods for getting the display aspect ratio and the active format definition. The proxy control seen by the proxy application will internally query the handler for its current `VideoFormatControl`, get the requested data, and return that to the calling proxy application. It is important to note that each call to a getter method on the control proxy will ask the handler for its most recent instance of that Control. This prevents stale accesses from occurring after a Service selection changes the list of active Controls.

Registering listeners for Control events is more complicated. Given the architecture of the WBA libraries without intermediation by the OCAP stack, it is impossible to maintain a current event listener registered by a remote application, unless that proxy application re-adds its listener each time it receives a `ServiceContextEvent` indicating that a change of Service occurred.

When the proxy application adds an event listener to a Control, the WBA library will query the current instance of the Control and add an internally defined listener for events. This internal listener will then forward those events back to the proxy control which will call the proxy application's event listener interface.

6.7.2.5 Custom Controls

It is possible for an `OCSHaredVideoHandler` to add custom controls to the list of Controls passed to the Proxy. These custom controls may include more detailed program information or parental control information to other applications. It is required that the Proxy application have direct knowledge of those Controls in order to use them, as their names and interfaces are not otherwise directly discoverable. Detailed instructions for publishing custom controls through the shared video subsystem are described in the WBA API specification.

6.7.3 OCAP Shared Video Functionality

This section summarizes the functionality available directly from the shared video handler/proxy system.

6.7.3.1 Service Selection

The service presented on the shared video window can be controlled through the use of these methods in the `OCSHaredVideoProxy` class:

- `getService()`: allows the application to retrieve the current service.
- `selectService(Service)`: allows the application to reselect a previously stored service or a new service the application has created itself.
- `selectService(MediaLocator)`: selects a new service, based on a media locator.
- `selectLastService()`: reselects the previously selected service (as the "Last" button on the remote does).

6.7.3.2 Video Manipulation

The video playback can be manipulated through the following methods in the `OCSHaredVideoProxy` class:

- `setRate()`, `getRate()`: queries and sets the playback speed.

- `setMediaTime()`, `getMediaTime()`: allows the application to get/set the currently playing position of a recorded or on-demand stream.
- `scaleVideo()`, `restoreFullScreenVideo()`: manipulates the size and position of the video being displayed.
- `stop()`: stops the current stream.

