OpenCable Application Platform Specifications
OCAP Extensions

OCAP Digital Video Recorder (DVR)

OC-SP-OCAP-DVR-I04-071220

ISSUED

Notice

This OpenCable specification 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.

Document Status Sheet

Document Control Number:	OC-SP-OCAP-DVR-I04-071220
Document Title:	OCAP Digital Video Recorder (DVR)
Revision History:	I01 – Released 5/24/04
	I02 – Released 5/24/05
	I03 – Released 5/9/07
	I04 – Released 12/20/07
Date:	December 20, 2007
Status:	~~Work in Progress~~ ~~Draft~~ **Issued** ~~Closed~~
Distribution Restrictions:	~~Author Only~~ ~~CL/Member~~ ~~CL/ Member/ Vendor~~ **Public**

Key to Document Status Codes:

Work in Progress	An incomplete document, designed to guide discussion and generate feedback that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Trademarks:

Contents

Figures

Tables

This page intentionally left blank.

1   SCOPE

This document defines a minimal profile for a Digital Video Recorder (DVR) software environment for digital cable receivers with local storage, and is a modular extension to the OpenCable Application Platform 1.0 ([OCAP1.0]). The OCAP 1.0 Profile, and the OCAP DVR Profile (this document), were developed by Cable Television Laboratories, Inc. (CableLabs), in conjunction with representatives from its member cable operating companies, as well as leading software and hardware firms.

The OCAP DVR is based on the OCAP 1.0 Profile and includes that document in its entirety.

1.1   OCAP DVR Purpose

The OCAP DVR is an application interface that includes all required Application Program Interfaces (APIs), content and data formats, and protocols, up to the application level. Applications developed to the OCAP DVR will be executed on OpenCable-compliant host devices. The OCAP DVR allows cable operators to deploy their applications and services on all OpenCable-compliant host devices connected to their networks.

The OCAP DVR platform SHALL be applicable to a wide variety of hardware and operating systems to allow Consumer Electronics (CE) manufacturers flexibility in implementation. A primary objective in defining the OCAP DVR is to enable competing implementations of the OCAP DVR platform by CE manufacturers.

1.2   OCAP DVR Requirements

The OCAP DVR platform has been designed to meet specific requirements that are not commonly applied to other DVR environments. Some of these requirements are related to content protection obligations that cable operators encounter, such that broadcast event descriptions might be maintained by applications, and that the platform supports DVR applications deployed by various service providers that have no a priori knowledge of each other and may compete for resources.

1.3   OCAP DVR Application Areas (informative)

The information in this section is informative to the OCAP DVR.

This section identifies the applications and services that could be made available to the viewer when using an OCAP DVR-compliant terminal. The descriptions of the applications are intended to demonstrate the scope of services required from [OCAP1.0].

1.3.1   Personal Video Recorder (PVR)

A critical application enabled by this platform is a PVR. A PVR application may be an extension to an Electronic Program Guide (EPG) that enables viewers to select programming events to record, and to select recorded events for viewing. Event listing and selection may be integrated into the EPG. Future broadcast events may be selected for recording. For instance, on Thursday night, a viewer might choose to record Sunday's broadcast of '60 Minutes'. Once the recording is scheduled, the PVR application will record the event without further input from the viewer. A PVR might also allow users to schedule the recording of a package of events, such as a season of 'Friends'.

1.3.2   Time Shift

Another critical application can be called 'time shift'. This set of features allows viewers to record the currently selected broadcast event, pause the current event, and rewind the current event. This capability is accomplished by

the use of a 'time-shift buffer', which temporarily stores the broadcast stream. If a user selects to pause the live broadcast, the current frame of video is displayed, while the ongoing broadcast is still saved to the time-shift buffer. The contents of the buffer, up to the beginning of the current event, can be saved in permanent storage, and the contents of the time-shift buffer can be viewed with 'trick-play' modes, such as rewind and fast-forward.

1.3.3   Pushed Content

Applications may schedule recording or perform immediate recording of broadcast events with initiation by a viewer. Many services and applications may use this capability to provide promotional material or targeted ads to viewers. These applications use the same API features as PVR applications.

2   REFERENCES

This section provides the normative and informative references used to create this profile.

2.1   Normative References

Note: Information contained in these normative references is required for all implementations. Notwithstanding, intellectual property rights may be required to use or implement these normative references.

The following documents contain provisions which, through reference in this text, constitute provisions of the present document. References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific:

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies.

The following table lists the normative references for this Profile: [1]

References	Edition	Description
[OC-SEC]		OpenCable System Security Specification, OC-SP-SEC-I07-061031, October 31, 2006, Cable Television Laboratories, Inc.
[OCAP1.0]	1.0	OpenCable Application Platform Specification, OCAP 1.0 Profile, OC-SP-OCAP1.0.1-070824, August 24, 2007, Cable Television Laboratories, Inc.
[HOST2.0]		OpenCable Host Device 2.0 Core Functional Requirements, OC-SP-HOST2.0-CFR-C01-071113, November 13, 2007, Cable Television Laboratories, Inc.
[HOST2-DVR]		OpenCable Host 2.0 DVR Extension, OC-SP-HOST2-DVREXT-I01-050502, May 2, 2005, Cable Television Laboratories, Inc.
[FIPS-46-3]	99 Oct 25	Data Encryption Standard (DES)
[FIPS-140-2]	01 May 25	Security Requirements for Cryptographic Modules
[FIPS-197]	2001 November 26	Advanced Encryption Standard (AES)
[A088 Rev. 2] MHP PVR/PDR Common Core	2006 September	Digital Video Broadcasting (DVB); Digital Recording Extension to Globally Executable MHP (GEM)
[DVB-GEM 1.0.2]	ETSI TS 102 819 v1.3.1, October 2005	Digital Video Broadcasting (DVB) Globally Executable MHP version 1.0.2

---

[1] Revised Normative References per ECN OCAP-DVR-N-05.0805-5 by GO on 12/20/05, OCAP-DVR-N-07.1017-1 by kb 4/24/07, OCAP-DVR-N-07.1084-1 by kb 11/19/07.

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

- European Telecommunications Standards Institute (ETSI), www.etsi.org

3   DEFINITIONS AND ABBREVIATIONS

3.1   Definitions

Digital Video Recorder (DVR)	A hardware/software platform that enables viewers to store digital video content. DVR systems enable applications such as PVRs.
Personal Video Recorder (PVR)	An application that enables viewers to schedule the recording of broadcast events, and to view and display previously recorded events.
Time-Shift	A set of functionality that enables viewers to record, pause, and rewind/fast-forward through a real-time broadcast event.
Time-Shift Buffer	A portion of memory that enables Time-Shift functionality.
Transrating	A process of modifying MPEG compression to achieve greater compression.

3.2   Abbreviations

BWP	Buffering Without Presentation
DVR	Digital Video Recorder
PVR	Personal Video Recorder
TSB	Time-Shift Buffer

4   CONVENTIONS

The following conventions are used in this specification:

- The Courier New font type is used to indicate code examples, names of properties, and other information that MUST be entered exactly as-is: `code example font`

- **Boldfaced** text is used as emphasis.

4.1   Profile Language

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"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 there may exist valid reasons 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.

4.2   Organization

This document uses the OpenCable Application Platform Specification, OCAP 1.0 Profile [OCAP1.0] as its base. Where applicable, OCAP DVR sections reference the corresponding section within [OCAP1.0].

The OCAP DVR Profile adds packages to the OCAP 1.0 Profile, as seen in Annex D, OCAP DVR API (org.ocap.dvr); Annex E, OCAP DVR Storage API (org.ocap.dvr.storage); OCAP Shared DVR API (org.ocap.shared.dvr) - see [A088 Rev. 2]; OCAP Shared DVR Navigation API (org.ocap.shared.dvr.navigation) - see [A088 Rev. 2]; OCAP Shared Media API (org.ocap.shared.media) - see [A088 Rev. 2]; Annex J, OCAP DVR Event API (org.ocap.dvr.event)

5 GENERAL CONSIDERATIONS

5.1 Introduction

This Profile fully defines a DVR extension to [OCAP1.0].This Profile defines a platform to enable applications to record and playback broadcast events, and to time-shift broadcast events. These functions are considered basic capabilities of a DVR platform, and the platform is limited in scope to support these basic functions. Advanced functions, such as preference engines or targeted content, are not explicitly supported by this platform and are considered application level functions.

5.2 Relationship with OCAP and GEM Specifications

Implementers of this Profile SHALL also fully implement [OCAP1.0]. [OCAP1.0] and this DVR platform are related in such a way that [OCAP1.0] implementations have no build-time or runtime dependencies on the DVR platform, while OCAP DVR implementations depend on the full implementation of [OCAP1.0].

With exceptions noted elsewhere in this document, all normative clauses of MHP PVR/PDR Common Core Specification [A088 Rev. 2], Digital Video Broadcasting (DVB); Digital Recording Extension to Globally Executable MHP, SHALL apply.

5.3 Basic Architecture (Informative)

Figure 5–1 shows an overview of the architecture assumed by the present document. Several aspects are omitted for the sake of clarity.
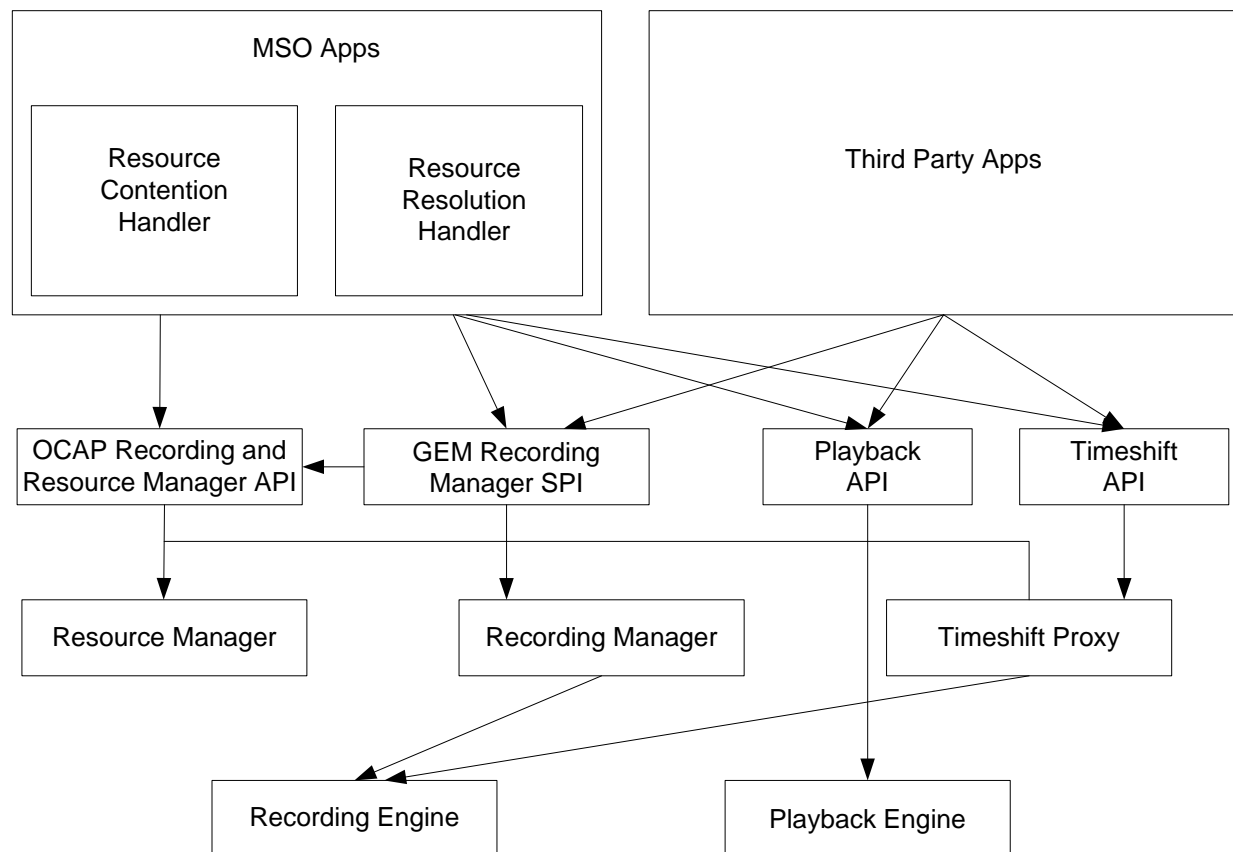
Figure 5–1 - DVR Architecture

In this architecture, among other things, the recording manager is responsible for:

- Managing pending recording requests.

- Interfacing with the resource manager to reserve resources required for pending and current recordings.

- Updating the conflict status of pending recording requests based on the resource availability.

- Interfacing with the recording engine to start and stop recordings at appropriate times.

- Updating the status of recording requests and creating `RecordedServices` as recordings are made.

Resource manager in this architecture is an extension of the resource manager functionality required to implement the OCAP 1.0 specification. Among other things, the resource manager is responsible for:

- Detecting resource conflicts between pending recording requests.

- Detecting resource conflicts between pending and in-progress recording requests.

- Detecting resource conflicts between recording requests and other activities that require resource reservations, such as time-shift buffering, presentation of broadcast services by services contexts, the `NetworkInterfaceController` reserve method, etc.

- Notifying the resource contention handler when the resource conflicts are detected.

- Resolving conflicts based on the priority specified by MSO applications.

In this architecture, an MSO application may register to handle resource contentions. When the resource manager detects a resource contention, the contention handler is invoked to resolve the resource contention.

An MSO application may register to handle request resolution. Typically this would be an application that has access to electronic program guide (EPG) listing data. When the recording manager encounters a recording request that requires additional information to process, this request resolution handler is invoked. The request resolution handler may process series requests and schedule recording requests corresponding to individual episodes.

Applications may use the GEM recording manager API to:

- Request recordings to be made,

- Manage the list of recording requests maintained by the recording manager,

- Manage recorded services.

5.4   API Support Property[2]

OCAP Hosts that support the OCAP DVR extension SHALL indicate this support with the system properties as defined in section 13.3.12.2 of [OCAP1.0], per Table 5–1.

Table 5–1 - API Support Property

Property	Description	Value	Application Access
ocap.api.option.dvr	System property indicating that OCAP DVR extension is supported by the Host device.	"1.0"	signed and unsigned

---

[2]Added this section per ECN OCAP-DVR-N-05.0842-1 by GO on 12/21/05; Revised Table 5-1 per ECN OCAP-DVR-N-06.0863-1 by GO on 9/25/06.

6 RECORDING AND PLAYBACK PROCESS

This chapter describes the recording and playback process for scheduled recordings and time-shift recording.

6.1 DVB-GEM Specification Correspondence

This section corresponds to [A088 Rev. 2], Chapter 6 as follows: [3]

Table 6–1 - Correlation between OCAP 1.0 and [A088 Rev. 2] MHP PVR/PDR Common Core Specification

OCAP Compliance	MHP PVR/PDR Common Core Specification Section	GEM Compliance
6, Recording and Playback Process	6 Recording and Playback Process	Extension
6.2.1.1, Scheduled Recordings	No corresponding section	Extension
6.2.1.1.1, RecordingSpecs	No corresponding section	Extension
6.2.1.1.2, Managing Recording Requests	6.1 Managing scheduled recording	Extension
6.2.1.1.3, The Recording Process	6.2 The recording process	Extension
6.2.1.1.3.1, Identifying Streams to be Recorded	No corresponding section	Extension
6.2.1.1.3.2, Identifying and Recording Applications	No corresponding section	Extension
6.2.1.1.4, Managing Completed Recordings	6.3 Managing completed recordings	Extension
6.2.1.1.5, Recording Request State Transitions (Informative)	No corresponding section	Extension
6.2.1.1.6, Resource Management for Recording Requests	No corresponding section	Extension
6.2.1.1.7, Request Resolution Process	No corresponding section	Extension
6.2.1.1.8, RecordedService	No corresponding section	Extension
6.2.1.2, Playback of Recorded Services	6.4 Playback of scheduled recordings	Extension
6.2.1.2.1, Process for Playback	6.4.1 Process of playback	Extension
6.2.1.2.1, Process for Playback	6.4.2 Event during playback	Extension
6.2.1.3, Time-Shift Buffer	6.5 Time-shift	Extension
6.2.1.3.1, Overview	No corresponding section	Extension
6.2.1.3.2, Recording	6.5.1 Recording	Extension
6.2.1.3.2.1, Identifying the Streams to be Recorded	No corresponding section	Extension
6.2.1.3.2.2, Identifying and Recording Applications	No corresponding section	Extension
6.2.1.3.3, Playback	6.5.2 Playback	Extension
6.2.1.3.4, Resource Management	No corresponding section	Extension
6.2.1.4, Storage	No corresponding section	Extension
6.2.1.4.1, Storage Management	No corresponding section	Extension
6.2.1.4.2, Storage Initialization	No corresponding section	Extension
6.2.1.5, Lightweight Triggers	No corresponding section	Extension

---

[3]Revised Table 6–1 per ECN OCAP-DVR-N-06.0898-3 by GO on 9/25/06.

6.2   OCAP 1.0 DVR Specific Requirements

6.2.1   Extensions to [A088 Rev. 2] MHP PVR/PDR Common Core

6.2.1.1   *Scheduled Recordings*

6.2.1.1.1   *RecordingSpecs*[4]

Applications may use an extension of the `RecordingSpec` class to specify a recording request. Implementations SHALL support the following extensions of `RecordingSpec` passed in as an argument to the `RecordingManager.record` method:

- org.ocap.shared.dvr.LocatorRecordingSpec

- org.ocap.shared.dvr.ServiceRecordingSpec

- org.ocap.shared.dvr.ServiceContextRecordingSpec

- org.ocap.dvr.PrivateRecordingSpec

Implementations SHALL support `RecordingProperties` specified through the extension `OcapRecordingProperties`.

If a `LocatorRecordingSpec`, `ServiceRecordingSpec` or `ServiceContextRecordingSpec` object is specified in the source parameter in a `record(RecordingRequest source)` method, a `LeafRecordingRequest` object returns. If a `PrivateRecordingSpec` object is specified in the source parameter in a `record(RecordingRequest source)` method, a `ParentRecordingRequest` object returns, according to Section 6.2.1.1.7, Request Resolution Process.

6.2.1.1.2   *Managing Recording Requests*[5]

In addition to the activities defined in clause 6.1 of [A088 Rev. 2], the process of managing recording requests SHALL include the following:

a) Detecting conflicts between multiple recording requests and between recording requests and resource reservations for other activities in the system. When conflicts are detected, the conflicts are resolved as specified in Section 6.2.1.1.6, Resource Management for Recording Requests8F, for recording requests. After the conflict is resolved, status of the recording request SHALL be updated based on the conflict resolution. Recording requests that are not expected to be recorded due to resource conflicts SHOULD NOT be deleted by the implementation unless explicitly deleted by the application. When a recording is scheduled or rescheduled, the recording database SHALL be checked for concurrent resource usages. If it is determined that the recording being scheduled will cause a point in time where there are more ongoing recordings than available resources, then the implementation SHALL resolve this conflict, as described in Section 6.2.1.1.6, Resource Management for Recording Requests8F, and invoke the `ResourceContentionHandler`, if one is registered.

As a result of this processing, the recording being scheduled, and any recordings that overlap this recording, will result in either the `PENDING_NO_CONFLICT_STATE` or `PENDING_WITH_CONFLICT_STATE`, excluding `IN_PROGRESS` recordings, which will retain their state, and where:

- The `PENDING_NO_CONFLICT_STATE` indicates that the recording has been granted a reservation for the usage of the needed resources, and that it is expected to complete in its entirety.

---

[4]Revised this subsection per ECN OCAP-DVR-N-06.0903-1 by GO on 9/25/06.

[5]Revised this subsection per ECN OCAP-DVR-N-05.0830-2 by GO on 12/19/05.

- The PENDING_WITH_CONFLICT_STATE indicates that at some point during the recording there exists too many simultaneous scheduled usages of the needed resources, and that other recordings have been given priority over this recording, and therefore, this recording is not expected to complete.

The priority property field, which can contain the values RECORD_IF_NO_CONFLICT and RECORD_WITH_CONFLICT, is only referenced at the time that the recording is scheduled to start. At that time, the recording attempting to start will already have been placed in one of the pending states, as described immediately above.

If, at recording start time, the recording is in the PENDING_WITH_CONFLICT_STATE, and the priority is set to RECORD_IF_NO_CONFLICTS, no attempts to acquire resources or physically start the recording will occur, and the recording SHALL be immediately marked as FAILED.

If, at recording start time, the recording is in the PENDING_WITH_CONFLICT_STATE and the priority is set to RECORD_WITH_CONFLICTS, or the recording is in the PENDING_NO_CONFLICTS_STATE, the implementation SHALL attempt to start the recording by acquiring the needed resources. This may also cause invocation of a registered ResourceContentionHandler, since non-DVR resource usages may be in progress.

b) Invoking the Request resolution handler when the record method is called with an instance of PrivateRecordingSpec. Details of how the request resolution process SHOULD be handled is defined in Section 6.2.1.1.7, Request Resolution Process.

c) The effect of storage device detachment and reattachment on scheduling of new recordings, pending recordings, and recording initiation SHALL be handled as detailed in Section 6.2.1.4.3.[6]

6.2.1.1.3   The Recording Process[7]

The recording process as defined in clause 6.2 of [A088 Rev. 2], SHALL be followed.

In addition to the activities defined in clause 6.3 of [A088 Rev. 2], the process for managing recordings SHALL include the following activity:

If the implementation estimates that a recording request that is in progress might not complete successfully due to lack of storage space available, the implementation SHOULD update the status of the recording in progress to reflect that. The estimation need not be accurate, and any proprietary algorithm MAY be used for this computation. Applications SHOULD use this only as an advisory notification and MAY use its own mechanisms to decide whether sufficient space is available to complete the recording.

If a ServiceContextRecordingSpec is used as the parameter to the record method, the implementation SHALL store and record the relevant portion already contained within any time-shift buffer associated with the service context, when the startTime is in the past.

Recordings initiated with the invocation of the record method with a ServiceContextRecordingSpec as the parameter SHALL be terminated if the service context is destroyed, or if an application selects another service on the service context.

If a ServiceRecordingSpec is used as the parameter to the record method, the implementation SHALL store and record relevant portions of the service already contained in any time shift buffer when the startTime is in the past. If a LocatorRecordingSpec is used, the implementation SHALL store and record relevant portions of the selected components of the service already contained in any time-shift buffer when the startTime is in the past. If the service is being buffered by more than one time-shift buffer, the one which contains the most amount of the

---

[6] Added per ECN OCAP-DVR-N-07.1076-2 by kb 11/19/07.

[7]Revised this section per ECN OCAP-DVR-N-05.0817-8 by GO on 9/25/06.

relevant portions of the service SHALL be used. In both these cases, if end time of the recording is in the future, the recording SHALL continue using the resources that were used for the time-shift buffering if allowed by resource contention handling. If the end time of the recording is in the past, an `IllegalArgumentException` is thrown if no relevant part of the service is contained in any time-shift buffer.

When content from a time-shift buffer is stored as a part of a recording, the `RecordedService.getRecordingStartTime()` method would return the start time at which the relevant portion of the content was originally recorded in the TSB.

The effect of storage device detachment and reattachment on in-progress recordings SHALL be handled as defined in Section 6.2.1.4.3.

6.2.1.1.3.1　　Identifying Streams to be Recorded

If the recording request was specified using a `ServiceRecordingSpec` or a `ServiceContextRecordingSpec`, the implementation SHALL:

　　a)　Record the audio and video streams that are present on the Service up to the limits in the recording capability of the OCAP-DVR device. NOTE: Minimum capabilities for recording streams are defined in Section 11.2.1.4, Recording Multiple Streams from the Same Service.

　　b)　Record Closed Captioning information and Content Advisory information, if these are included in the Service.

　　c)　Manage (increment where needed) CCI bits and store them along with the stream if CCI bits are present in the broadcast stream.

　　d)　Create a `TimeLine` accessible through the `TimeLineControl` API corresponding to each time-base associated with the Service as signaled through DSMCC `NPTReferenceDescriptors`.

Note:　　Recorded streams may be stored in a proprietary format in the OCAP DVR device.

6.2.1.1.3.2　　Identifying and Recording Applications

If an application is signaled as to be recorded (the scheduled recording flag in the application recording descriptor is set to '1'), and if the application does not rely on the use of dynamic data during its execution (`dynamic_flag` in the application recording descriptor is set to 0), then the application SHALL be recorded. Implementations MAY record all applications. Implementations MAY record dynamic data associated with the applications and make them available through data access APIs in a manner consistent with access of data from broadcast streams.

6.2.1.1.4　Managing Completed Recordings[8]

In addition to the activities defined in clause 6.3 of [A088 Rev. 2], the process for managing completed recordings SHALL include the following activities:

a)　Deleting the `RecordedService` once the expiration period is past, or space is needed for additional recordings, as indicated by the `OcapRecordingProperties` `retentionPriority`.

　　(1)　If the `retentionPriority` is `DELETE_AT_EXPIRATION`, then the implementation SHALL NOT allow any application to access the `RecordedService` past the expiration period. If playback is in progress when the recordings expiration period is reached, the playback SHALL be terminated. The implementation SHALL delete the `RecordedService` within one hour after the expiration of

---

[8]Revised this section per ECN OCAP-DVR-N-05.0806-3 and OCAP-DVR-N-05.0829-1 by GO on 11/4/05 and 12/19/05.

the recording. The implementation SHALL NOT allow playback of a recording marked for deletion.

(2) If the `retentionPriority` is a value other than `DELETE_AT_EXPIRATION`, then the implementation SHALL defer deletion of the `RecordedService` until the storage space is needed for other purposes. Recordings with lower values for `retentionPriority` SHALL be deleted before recordings with higher values for `retentionPriority`. When recordings have the same `retentionPriority`, older recordings, as determined by the `RecordedService` `getRecordingStartTime()`, SHALL be deleted first. Recordings with a `retentionPriority` value other than `DELETE_AT_EXPIRATION` SHALL remain accessible for playback and SHALL NOT be deleted while playback is in progress.

b) Maintaining all `OcapRecordingRequests` regardless of state until explicitly deleted by an application. This includes `OcapRecordingRequests` that are in the `CANCELLED_STATE`, `FAILED_STATE` and `DELETED_STATE`.

c) Implicitly deleting and reconstructing completed recordings due to storage device detachment and reattachment as detailed in Section 6.2.1.4.3.[9]

This specification extends the javadoc for the `org.ocap.shared.dvr.RecordingRequest.delete` method identified in [A088 Rev. 2]. The `RecordingRequest.getID` method SHALL NOT throw an `IllegalStateException` when called on a `RecordingRequest` reference delivered to a `RecordingChangedListener` after the request has been deleted from the database, and SHALL return the correct identifier for the deleted request.

6.2.1.1.5   Recording Request State Transitions (Informative)

The following diagram illustrates possible transitions between various states of the recording request:[10]

---

[9] Added per ECN OCAP-DVR-N-07.1076-2 by kb 11/19/07.

[10] Revised this figure per ECN OCAP-DVR-N-05.0806-3 by GO on 11/4/05.
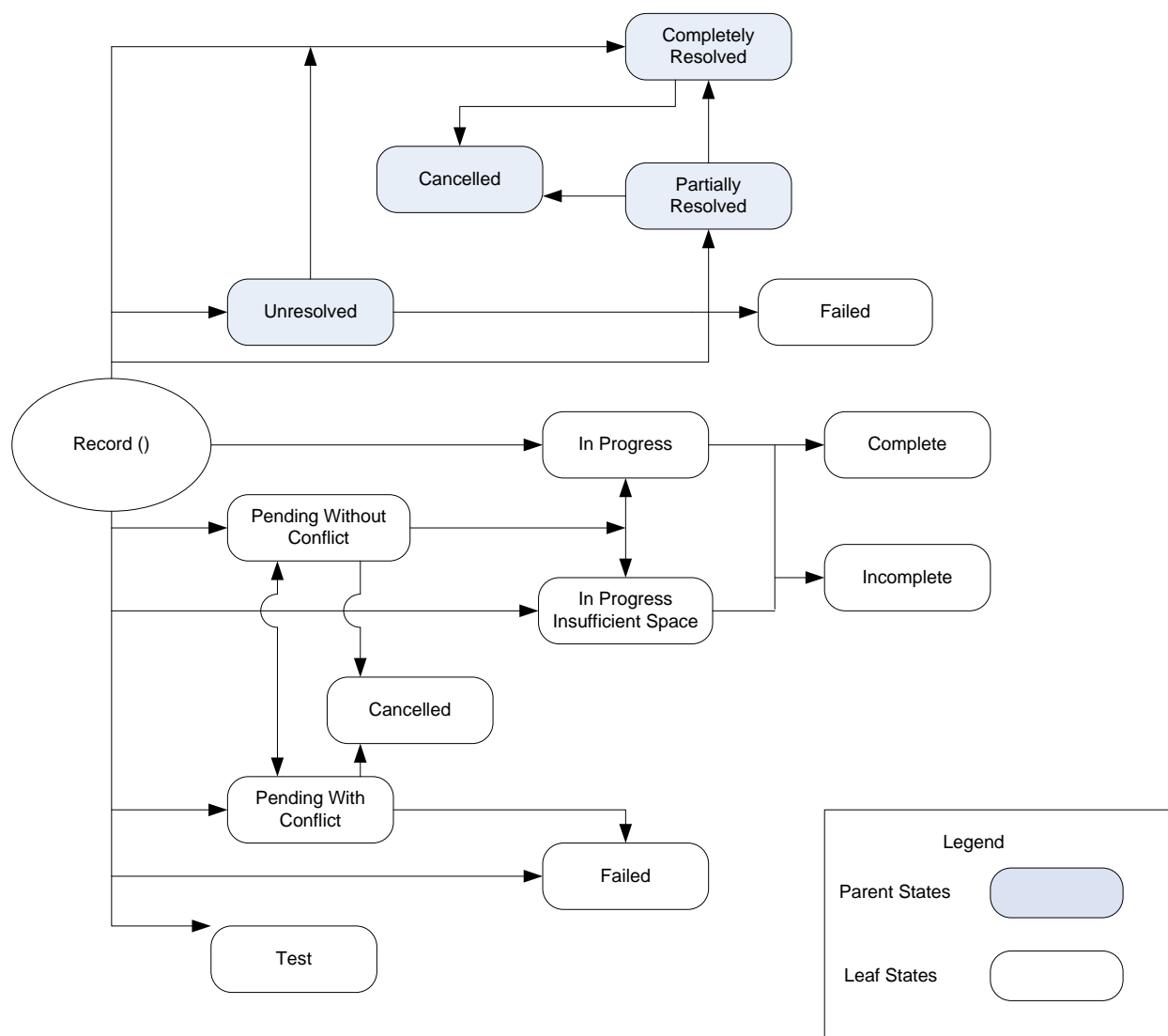
Figure 6–1 - Recording Request State Diagram

6.2.1.1.6   Resource Management for Recording Requests[11]

The implementation SHALL perform the following resource management activities for recording requests:

a) Create an instance of `RecordingResourceUsage` corresponding to each recording request in one of the pending states. The method `getResource(..)` MAY return `null` for `RecordingResourceUsages` corresponding to pending recording request.

b) When new recording requests are inserted, or when existing recording requests are modified, detect conflicts by checking if resources are sufficient to complete all recording requests in pending-without-conflict state and in-progress states. If conflicts are detected, resolve the conflict as specified in section 19.2.1.1 of [OCAP1.0]. Update the states of pending recording requests based on the conflict resolution. If the resolution of conflict resulted in a recording request in-progress state losing resources needed to continue recording, the recording SHALL be terminated and the state for the recording request SHALL be changed to incomplete state.

---

[11]Revised this section per ECN OCAP-DVR-N-05.0817-8 by GO on 9/25/06.

c) Before starting a recording, create an instance of `RecordingResourceUsage` corresponding to the recording request and reserve the resources required for the recording. The method `getResource(..)` for this recording resource usage SHALL NOT return `null` for any resource names returned by the method `getResourceNames()`.

d) Save the values of the requesting application's AppID and priority with each recording request. These values SHALL be used for any resource contention in which the recording request is involved, even if the requesting application is no longer running when the contention occurs. The stored priority SHALL NOT be affected by changes in the requesting application's priority after the recording request is made.

There is no requirement for implementations to maintain a history of prioritized `ResourceUsage` arrays as would be returned from the `ResourceContentionHandler.resolveResourceContention` or `OcapRecordingManager.setPrioritization` methods.

When a resource contention handler is registered and a recording is cancelled or deleted, the implementation SHALL NOT call the `ResourceContentionHandler.resolveResourceContention` method. If removal of a recording results in resource availability that allows other recordings to move from in-conflict to no-conflict, the choice of which recording gets moved is implementation-dependent.

Recursive calls to the `ResourceContentionHandler.resolveResourceContention` method can occur if resources are used by that method. `ResourceContentionHandler` designers SHOULD be aware of this possibility. It is recommended that the only resources used by this method are for graphics needed to display a prompt for consumer resolution of a conflict that is unresolvable by the `ResourceContentionHandler`.

6.2.1.1.7   Request Resolution Process

When recording requests are specified by applications using `PrivateRecordingSpec`, the implementation SHALL invoke the request resolution handler to resolve the request. The request resolution process includes the following:

a) When the `RecordingManager.record()` method is called with a `PrivateRecordingSpec` as an argument, the implementation SHALL create a `ParentRecordingRequest` in `UNRESOLVED` state. The implementation SHALL invoke any registered request resolution handler with the newly-created recording request as the parameter.

b) When the `OcapRecordingManager.resolve(..)` method is called with a `PrivateRecordingSpec` as an argument, the implementation SHALL create a `ParentRecordingRequest` in `UNRESOLVED` state and make that a child of the recording request that was passed as an argument to the `resolve(..)` method. The implementation SHALL invoke any registered request resolution handler with the newly-created recording request as an argument.

c) When the `OcapRecordingManager.resolve(..)` method is called with a `RecordingSpec` other than the `PrivateRecordingSpec` as an argument, the implementation SHALL create a `LeafRecordingRequest` and make that a child of the recording request that was passed as an argument to the `resolve(..)` method.

6.2.1.1.8   RecordedService [12]

The implementation SHALL create a `RecordedService` when a recording request enters one of the 'in progress' states. A `RecordedService` SHALL NOT be listed in `ServiceLists` returned by the `SIManager` class in

---

[12]Revised this subsection per ECN OCAP-DVR-N-06.0892-1 and OCAP-DVR-N-06.0897-4 by GO on 9/25/06.

the JavaTV service package. The `getLocator()` method of `RecordedService` SHALL return an OCAP Locator that is different from the originating service Locator. The `getName()` method for a recorded service SHALL return a unique name starting with the string "RecordedService". The method `getServiceType()` SHALL return `RECORDED_SERVICE`. The method `hasMultipleInstance()` SHALL always return `false`. The implementation SHALL support the `retrieveDetails(SIRequester requester)` method; however, it is optional for the host to return a valid `ServiceDetails` object, i.e., if the host doesn't have a capability of recording necessary information that will be contained in a `ServiceDetails` object, an `SIRequestor.notifyFailure(SIRequestFailureType)` method will be called with `DATA_UNAVAILABLE` reason. If the host has capability of recording information to create a `ServiceDetails`, the host SHALL reflect change of elementary stream in the recorded service as follows. If a `RecordedService` is not playing back when the `retrieveDetails(SIRequester requester)` method is called, the object implementing the `ServiceDetails` SHALL contain information at the beginning of the recorded content in `RecordedService`. In this case, even if the `RecordedService` has been played once and stopped in the middle of the content, the object implementing `ServiceDetails` SHALL contain information at the beginning of the recorded content. On the other hand, if a `RecordedService` is playing back when the `retrieveDetails(SIRequester requester)` method is called, the `ServiceDetails` SHALL contain information at the current playback point of the recorded content. Whichever application has started playback, set a rate or changed the current position of a `RecordedService`, and whichever application calls the `retrieveDetails(SIRequester requester)` method, information in a `ServiceDetails` object is information at the current playback point. All methods in objects implementing `ServiceDetails` for `RecordedServices` SHOULD function the same as objects implementing the `ServiceDetails` for broadcast services.

If multiple `ServiceContexts` are presenting a single `RecordedService`, an `SIRequestor.notifySuccess(SIRetrievable[] result)` method returns a set of `ServiceDetails` objects that corresponds to those `ServiceContexts`, i.e., one `ServiceDetails` object for each `ServiceContext`. Note that an OCAP-J application can't identify `ServiceDetails` for each `ServiceContext` in this case; however, this is not a significant restriction since this is very rare case.

6.2.1.2   *Playback of Recorded Services*

6.2.1.2.1   *Process for Playback[13]*

The process for playback SHALL be as defined in clause 6.4 of [A088 Rev. 2] and additionally constrained as follows:

  a) When playing content that is currently being recorded, if the end of the content is reached and recording stops, the playback must continue without interruption at this point, regardless of any (implementation-dependent) process to copy the newly-recorded content from any temporary buffer to a more permanent location on the storage device.

  b) When a recorded service is selected on a service context, the playback SHALL begin from the media time set using the method `RecordedService.setMediaTime(..)`.

  c) If the playback location is the same as the recording point (playing back the live point), the implementation SHALL display the broadcast stream rather than the stream coming off the storage medium.

The following rules apply to players presenting recorded content, both associated with a service context, presenting a recorded service, and those directly created from a `MediaLocator`:

  a) The method `setMediaTime(..)` called with a value of `Time` corresponding to `POSITIVE_INFINITY`

---

[13]Revised this section per ECN OCAP-DVR-N-06.0862-2 by GO on 9/25/06.

SHALL set the playback location to the current record point if the recording is still on-going, or to the end of the recording.

b) The method `setRate(0.0)` SHALL pause the playback displaying the last frame displayed. Any following `setRate` called with a non-zero parameter SHALL resume the playback at the specified rate from the paused location.

c) Audio SHALL NOT be presented for any playback rate other than 1.0.

d) If recording is ongoing, and if the playback (at a rate more than 1.0) hits the end of the recorded content, the implementation SHALL set the playback rate to 1.0 and send an `EndOfContentEvent` to any registered controller listeners, and doesn't send an `EndOfMediaEvent`. If the recording is not ongoing, and if the playback (at a rate more than 0.0) hits the end of the recorded content, the implementation SHALL set the playback rate to 0.0 and send an `EndOfContentEvent` to any registered controller listeners, and doesn't send an `EndOfMediaEvent`. Note that playback at a rate equal to 0.0 is different from stop of playback.[14]

e) If the player hits the beginning of media during playback at a rate less than 0.0, the implementation SHALL change the playback rate to 1.0 and send a `BeginningOfContentEvent` to any registered controller listeners.

f) The implementation SHALL set a playback rate to 1.0 for recorded contents that are presented at a current rate other than 1.0, if an OCAP-J application that set the current rate to the recorded contents is terminated by AIT/XAIT signaling (`application_control_code` or `trick_mode_aware_flag`) or any error case.[15]

g) If the current playback rate is 1.0, a `setRate()` call by any application will be successful unless there are errors. If the current playback rate is not 1.0, only a `setRate()` call that satisfies either of the following conditions (1) or (2), will succeed:

   (1) The caller belongs to the same service as the application that has set the current playback rate, and the caller has an equal or higher application priority value than the application that set the current playback rate. Note that the services that two applications belong to are identified by a service_id (source_id) instead of a service instance.

   (2) The caller application instance is the same application instance that has set the current playback rate.

h) If the current playback rate is 1.0, a `stop()/syncStart()/setTimeBase()/setStopTime()/setMediaTime()` call by any application will be successful unless there are errors. If the current playback rate is not 1.0, only a `stop()/syncStart()/setTimeBase()/setStopTime()/setMediaTime()` call that satisfies either of the following conditions (1) or (2), will succeed:

   (1) The `stop()/syncStart()/setTimeBase()/setStopTime()/setMediaTime()` caller belongs to a same service as the application that has set the current playback rate by a `setRate()` call, and the `stop()/syncStart()/setTimeBase()/setStopTime()/setMediaTime()` caller has an equal or higher application priority value than the application that set the current playback rate by a `setRate()` call. Note that the services that two applications belong to are identified by a service_id (source_id) instead of a service instance.

---

[14]Revised step d) per ECN OCAP-DVR-N-06.0898-3 by GO on 9/25/06.

[15]Added this statement per ECN OCAP-DVR-N-05.0819-1 by GO on 11/4/05.

(2) The `stop()`/`syncStart()`/`setTimeBase()`/`setStopTime()`/`setMediaTime()` caller
application instance is the same application instance that has set the current playback rate by a
`setRate()` call.

Closed Caption and Content Advisory data as originally delivered from the network SHALL be present on
playback.

6.2.1.3   Time-Shift Buffer[16]

6.2.1.3.1   Overview

Time-shifting is the feature that enables an application to pause and rewind a service that is being broadcast. The
implementation of this feature is split into two parts - the time-shift buffering and time-shifted presentation.

A time-shift buffer is used to store a finite amount of live broadcast in order to apply trick-mode controls and
instantaneous recording of a presenting service, or a buffering without presentation request.

The `org.ocap.dvr.TimeShiftProperties` interface SHALL be implemented by any class that also
implements `javax.tv.service.selection.ServiceContext` when the Host device supports the OCAP
DVR option. Time-shifted presentation for a service context is enabled and controlled by applications by setting the
value for the minimum time-shift duration for the service context. The method `setMinimumDuration(..)` in
the `TimeShiftProperties` interface is used for setting the minimum duration content will be buffered for a
service context. If time-shift presentation is enabled for a service context, any broadcast service presented on the
service context SHALL also be simultaneously buffered or recorded so that an application MAY rewind the
playback location till the buffer depth of the time-shift buffer is reached. The implementation SHALL implicitly
associate the service context to one or more time-shift buffers or one or more recordings and use the content stored
in the time-shift buffers or the recordings to facilitate time-shifted presentation on that service context. When a
service context is associated with time-shift buffers or recordings, each time-shift buffer or recording is said to be
"attached" to the service context.

The `TimeShiftProperties` interface also allows an application to control other preferences, including a
preference to retain time-shift contents when a new service is selected, and a preference to buffer the last service.
Retaining time-shift content during service selection SHALL NOT affect a Host device's ability to comply with
service selection performance requirements. Host devices that do not have hardware resources to implement this
preference without affecting service selection performance compliance MAY ignore it.

6.2.1.3.1.1   Time-shift Buffering Without Presentation

In addition to time shift of a service context, applications MAY request buffering of services not selected on a
service context and that are not being presented to display or audio outputs. This is referred to as buffering without
presentation (BWP). Applications MAY request the implementation to start time-shift buffering of a service using
the `OcapRecordingManager.requestBuffering` method. The `requestBuffering` method takes a
`BufferRequest` parameter created by the calling application. The
`OcapRecordingManager.getBufferingRequests` method returns all active BWP requests.

BWP requests share resources with other resource requests. When a service context or scheduled recording begins
on the same service as a BWP request, the resources are shared. For example; if a service context is used to select a
service being buffered by a BWP request, the buffering resources become available to the service context, and time-
shifting in the past can be accomplished as soon as the select completes. If the service context is used to select a
different service later on, the implementation attempts to continue to honor the BWP request on the service
contained in the request. A BWP request is not attached to or associated with a `ServiceContext` or scheduled
recording except for sharing of resources when both are selected on the same service. A BWP request service is set

---

[16]Revised this section per ECN OCAP-DVR-N-05.0817-8 by GO on 9/25/06.

when the request is created and can be changed at any time by an application with access permissions to the BWP request.

A BWP request has a lifetime during which an implementation will make a "best effort" to honor it. A BWP request is honored when it is buffering on its own or when it is sharing resources with functions such as service presentation, a scheduled recording, or another BWP request. The implementation may need to cause a tune to honor a BWP request. When a BWP request is sharing resources with another function and that function ceases to share resources (e.g., tunes away), the implementation continues attempts to honor the request until an application cancels the request. Other aspects of BWP requests are stated below:

- BWP requests do not persist across reboots and power-cycles.

- Applications make requests for BWP requests that the implementation will honor if resources are available.

- A registered resource contention handler is used to resolve BWP request conflicts. `ResourceUsage` objects with the AppID of the application that created the request are used for contention handling. When an implementation creates a BWP request, any `ResourceUsage` for this request will contain a null AppID.

- Events are not generated for BWP status.

- Controlling a network interface to honor a BWP request may generate events as defined by OCAP, e.g., if a tune is performed.

BWP requests have the following attributes that can be set or queried by an application:

- Service - The Service to buffer.

- Minimum duration - The minimum duration that must be buffered for this request to be honored. Resource contention of BWP requests SHALL be based on this value. If the BWP request is sharing buffer resources with a function that has a greater duration, the value of that duration is used; otherwise the implementation attempts to honor the BWP request duration.

- Maximum duration - The maximum duration an application might need for this request. Informs the implementation as to how much storage is best to set aside for this request.

- Extended file access permissions - Determines which applications can change the attributes of a request.

- App Id - Application identifier of the application that created the request. Will be null if the implementation created the request. Read only.

The implementation SHALL handle a last-channel buffering request for a service context by setting a BWP request for the last channel selected for that context. Every time the service context is tuned to a new channel, the implementation changes the service in the last channel BWP request to follow the last channel. A BWP request representing a last channel buffering request SHALL contain the AppID of the application that set the last channel preference. The implementation can determine when a last channel request SHOULD change in an implementation-specific fashion. For example, an implementation may wait five seconds to avoid buffering a last channel while the consumer is rapidly changing channels.

6.2.1.3.2 Recording

The time-shift recording process as defined in clause 6.5 of [A088 Rev. 2] SHALL be followed. In addition to the activities defined in clause 6.5.1 of [A088 Rev. 2], the process for the time-shift recording SHALL include the following activity:

Time-shift resources contain content received since the time of a boot-up or the most recent flush event following the last boot-up.

6.2.1.3.2.1   Identifying the Streams to be Recorded

The implementation SHALL:

a) Record the audio and video streams that are present on the Service up to the limits in the recording capability of the OCAP-DVR device. NOTE: Minimum capabilities for recording streams are defined in Section 11.2.1.4, Recording Multiple Streams from the Same Service.

b) Record Closed Captioning information and Content Advisory information if these are included in the Service.

c) Manage (increment where needed) CCI bits and store them along with the stream, if CCI bits are present in the broadcast stream.

6.2.1.3.2.2   Identifying and Recording Applications

All applications bound to the broadcast service with `time_shift` flag in the application recording descriptor set to '1' SHALL be recorded. Implementations MAY record dynamic data associated with the applications and make them available through data access APIs in a manner consistent with access of data from broadcast streams.

6.2.1.3.3   Playback

The time-shift playback process as defined in clause 6.5.2 of [A088 Rev. 2] SHALL be followed.

As previously described, an application can enable or disable time-shift buffering of a service context using the `TimeShiftProperties.setMinimumDuration` method. Calling this method SHALL cause a time-shift resource in the form of one or more time-shift buffers or in-progress recordings to be implicitly attached to or detached from a service context, depending upon the value of the duration parameter passed to the method. However, the implementation SHALL ensure the time-shift resources allocated based on a call to the `setMinimumDuration` method are permanent. For example; a recording cannot be used to satisfy the call because it will not be usable once the current time minus the minimum duration value passes the end of the recording. The implementation SHALL adhere to the following rules to complete the process of a time-shift buffer or recording attaching to and detaching from a service context.

a) If an application enables time-shifted presentation for a service context that was presenting a broadcast service during a service selection, the implementation SHALL wait till the selection completes and implicitly attach an ongoing recording of the same service, or an available time-shift buffer to the service context, and use the recording or the time-shift buffer to facilitate time-shifted presentation of the service. The implementation SHALL begin the service presentation from the live point. If the time-shift buffer or recording used was recording the service selected, the implementation SHOULD make portions already recorded available to a controlling application based on the time-shift duration value set by the application. If the time-shift duration is reduced and a time-shift buffer is attached to the service context, the implementation MAY reduce the size of the time-shift buffer accordingly.

It is implementation-dependent whether the implementation uses a time-shift buffer or a recording or combinations of the two, to enable time-shifting of broadcast service presentation. If the implementation uses a combination of time-shift buffers and recordings, the transition across the buffers or recordings SHALL NOT affect the JMF media time, the TimeShiftControl behavior, or the rewind duration.

b) When a new service is selected on a service context for which the time-shifted presentation is enabled, the implementation SHALL execute the same behavior as specified in step a) immediately above for time-shift enabled. In addition, the implementation MAY detach any time-shift buffer or recording being used for the previous service and attach a different time-shift buffer or recording combination to the corresponding service context during service selection. If the `TimeShiftProperties.setLastServiceBuffered` method has been called with a value of `true,` the implementation SHALL continue buffering the previous

service if resources are available.

c) If an application disables time-shifted presentation for a service context that was presenting a broadcast service, any attached time-shift buffer or recording SHALL implicitly detach the service context. Once detached, a time-shift buffer is available for other uses. Disabling time-shifted presentation in a service context does not imply content flushing.

d) If a recording is initiated for a service being presented by a service context with an attached time-shift buffer, the implementation SHALL use the resources being used for time-shift buffering for the recording. The implementation SHALL not detach from any attached time-shift buffer when this occurs. In this case, a `RecordingTerminatedEvent` is not sent to any registered listeners of the `ServiceContext`. In addition, no resource contention is generated.

e) When a service recording completes or terminates, and the recording was attached to a service context as the time-shift resource, the implementation SHALL start time-shift buffering of the service on a different time-shift resource, if available. The implementation SHALL use a time-shift buffer if one is available. If not, a recording SHALL be used if available.

f) Implementations SHALL detach a recorded service attached to a service context once the attachment is not necessary for rewinding to the maximum depth as specified in the time-shift buffer properties.

g) A player associated with a service context with an attached time-shift buffer or recording, and that is presenting a broadcast service, SHALL follow all rules for a player associated with a service context presenting a recorded service; see Section 6.2.1.2. In addition, the following rules also apply to such a player:

   (1) If the write point of the time-shift buffer is about to overwrite the location corresponding to the current media time due to circular buffer reaching its depth, the implementation SHALL set the playback rate to 1.0 in order to prevent the location corresponding to the current media time being invalidated. The implementation SHALL send a `BeginningOfContentEvent` to any registered controller listeners. This SHOULD occur only when the playback is at a rate less than 1.0. The time-shift buffer implementation SHOULD make sure that the location corresponding to the current media time is never invalidated.

   (2) If the playback location is the same as the record point (playing back the live point), the implementation SHALL display the service with no delay from the broadcast service.

h) When a presentation loses time-shift resources due to resource contention or because an application disables time-shift, the implementation SHALL move the presentation to the live point, an `EnteringLiveModeEvent` SHALL be generated, and the `TimeShiftControl` SHALL be removed from the corresponding Player.

i) Default values for a `TimeShiftProperties` instance SHALL be set as follows:

   - Minimum time-shift duration to 0.
   - Last channel buffered to false.
   - Save time-shift contents on service selection to false.

6.2.1.3.4   Resource Management

Resource management for the time-shift buffer SHALL be done according to the following rules:

a) When an application requests the implementation to start buffering a broadcast service and if a time-shift buffer is available, the implementation SHALL create an instance of a `TimeShiftBufferResourceUsage` and attempt to allocate the resources needed for the buffering of the broadcast service. If there is a conflict, the conflict SHALL be resolved as specified in section 19.2.1.1 of [OCAP1.0].

b) If a time-shift buffer is attached to a service context, and if the resources needed to continue buffering are taken away from the time-shift buffer, and the resources are allocated to a recording that is not attached to the service context, the implementation SHALL:

    (1) Terminate buffering for the service context that has lost time-shift resources.

    (2) Continue presentation till the playback hits the end of the buffered content.

    (3) When playback hits the end of the buffered content, send a `RecordingTerminatedEvent` to any registered listeners on the service context.

    (4) If the application that selected the service in the service context owns resources for live broadcast presentation, the implementation SHALL jump the service presentation to the live point in the broadcast. Otherwise, the implementation SHALL stop the service context presentation and generate a `PresentationTerminatedEvent` with reason code `RESOURCES_REMOVED`.

c) When a time-shift buffer is attached to a service context, the resources used by the time-shift buffer SHALL be shared with the service context and SHALL be represented using a `SharedResourceUsage` if the shared resources are in a resource contention. The method `SharedResourceUsage.getResourceUsages()` SHALL return an instance of the `ServiceContextResourceUsage` and an instance of `TimeShiftBufferResourceUsage`.

d) When an ongoing recording is attached to a service context, or a recording request is recording the same service being presented by a service context with time-shifting disabled, the resources used by the recording SHALL be shared with the service context and SHALL be represented using a `SharedResourceUsage` if the shared resources are in a resource contention. The method `SharedResourceUsage.getResourceUsages()` SHALL return an instance of the `ServiceContextResourceUsage` and an instance of `RecordingResourceUsage`.

e) When a time-shift buffer is used to create a scheduled recording and the service is not being presented by a service context, the implementation SHALL create a `SharedResourceUsage` composed of a `TimeShiftBufferResourceUsage` and a `RecordingResourceUsage`.

f) When a time-shift buffer is used to create a scheduled recording and the service is being presented by a service context, the implementation SHALL create a `SharedResourceUsage` composed of a `TimeShiftBufferResourceUsage`, a `RecordingResourceUsage`, and a `ServiceContextResourceUsage`.

g) When time-shifted presentation is enabled on a service context, but the implementation cannot find a recording or time-shift buffer to attach to the service context, the implementation SHALL generate an `org.ocap.dvr.TimeShiftEvent` to the `ServiceContext` during a select method call or after presentation begins. The reason code SHALL be `NO_TIME_SHIFT_BUFFER`. Presentation SHALL continue without time-shift capabilities based on availability of other resources.

h) When time-shifted presentation is enabled on a service context and no time-shift buffer or recording was available but one becomes available during service presentation, the implementation SHALL generate an `org.ocap.dvr.TimeShiftEvent` to the service context. The reason code SHALL be `TIME_SHIFT_BUFFER_FOUND`. Time-shifted service presentation SHALL be available from that point in time. If more than one service context have time shift enabled and are awaiting resources, the contention SHALL be resolved as specified in clause a) of this section.

i) When time-shifted presentation is enabled on a service context and network signaling causes the time-shift properties to change, the implementation SHALL generate an `org.ocap.dvr.TimeShiftEvent` to the service context. The reason code SHALL be `TIME_SHIFT_PROPERTIES_CHANGED`. For instance, this could occur when the CCI bits dictate a limit on time-shift contents duration and the minimum duration is greater than that value.

j) When time-shifted presentation is disabled on a service context, if the application that selected the service in the service context owns resources for live broadcast presentation, the implementation SHALL jump the service presentation to the live point in the broadcast. Otherwise the implementation SHALL stop service context presentation and generate a `PresentationTerminatedEvent` with reason code `RESOURCES_REMOVED`.

k) When a presenting time-shift buffer or a recording are attached to a service context, any redundant resources SHALL be freed.

l) When a new service is selected on a service context with an attached ongoing recording, and the implementation cannot find another recording or a time-shift buffer to attach, the rules g) and h) SHALL apply.

Note: The resource management rules above allow for various time-shift designs. For example, systems that use a time-shift buffer to store a scheduled recording and convert it to a permanent recording can be implemented using these rules. In addition, it is recognized that a scheduled recording and a broadcast presentation of the same service MAY share the same hardware resources.

6.2.1.4   Storage

Storage devices are represented in OCAP by objects implementing the `org.ocap.storage.StorageProxy` interface. A `StorageProxy` may be extended by `StorageOption` interfaces that expose additional capabilities of a device. The `StorageProxy` interface also supports `LogicalStorageVolumes,` which allow applications to organize and control access to content. This Profile extends the base storage capabilities in OCAP 1.0 to support the storage and playback of full resolution video. Implementations MAY use storage architectures for DVR content that differ from a general purpose file system. For this reason, a new volume type, `MediaStorageVolume`, is introduced for this storage.

6.2.1.4.1   Storage Management[17]

The `MediaStorageOption` and `MediaStorageVolume` interfaces expose the special characteristics of DVR storage to applications. When a device is attached, the implementation SHALL determine whether it is supported for DVR media content storage. Implementations SHALL provide the `MediaStorageOption` via the `getOptions()` method on `StorageProxy` objects that are DVR media-capable.

The `MediaStorageOption` interface supports the creation of volumes for either DVR media storage or for general purpose file use. Volumes of the former type implement the `MediaStorageVolume` interface, which extends the `LogicalStorageVolume` interface and provides the ability to preallocate storage for a volume. Implementations SHALL support the creation of multiple instances of both `LogicalStorageVolume` and `MediaStorageVolume` on any `StorageProxy` that is capable of storing DVR content. Storage allocated for a `MediaStorageVolume` SHALL always be available for use by recordings created on that `MediaStorageVolume` until that storage is explicitly released by an application.

The first media storage volume that is created on a `StorageProxy` is the default recording volume and is used to record programs for which a destination `MediaStorageVolume` is not specified for a recording. If there are multiple media-capable `StorageProxy` objects available with default recording volumes, the implementation may choose any default recording volume as the destination for such a recording. An implementation SHALL NOT spread the content for a recording across multiple storage volumes. An implementation SHALL NOT resolve the default recording volume request to a specific MediaStorageVolume until recording is to be started.[18]

Implementations are not required to make files created for recordings on DVR media volumes visible through the java.io package. Implementations MAY allow applications to store and retrieve general purpose files on a

---

[17]Revised this section per ECN OCAP-DVR-N-06.0929-3 by GO on 12/8/06.
[18] Sentence added per ECN OCAP-DVR-N-07.1076-2 by kb 11/19/07.

MediaStorageVolume using the java.io package, but are not required to do so. Since media content is referenced through locators generated by the platform, an implementation MAY use multiple platform-specific files to represent a single media locator and MAY use storage architectures that differ significantly from typical file systems. Implementations SHALL support recordings of any length up to the available storage. The implementation SHALL ensure that the actual bandwidth available for DVR usage to StorageProxies always meets the combined maximum recording and playback capacity of the host device. The implementation SHALL delete any recordings, that, due to corruption, it can determine, cannot be played at all. The implementation SHALL delete any ancillary or index files related to a recording when a recording is lost or deleted.

A privileged application with MonitorAppPermission("storage") can block a MediaStorageVolume from access by any application. It does this by calling the MediaStorageVolume.removeAccess method and passing in a null parameter. When this occurs successfully, the MediaStorageVolume is effectively removed as an application-accessible system resource. This does not affect implicit time-shift buffering or buffering without presentation activities. MediaStorageVolume application access can be restored with the MediaStorageVolume.allowAccess method by passing in a null parameter. When restored, any recordings that were interrupted and are still in progress SHALL be segmented based on the recording segmentation definition for loss of resource as defined in [A088 Rev. 2].

6.2.1.4.2   Storage Initialization

Implementations that do not use a common storage architecture (e.g., file system type) for both general purpose files stored in LogicalStorageVolumes and media content stored in MediaStorageVolumes, may not be able to dynamically shift storage between one use and the other without destroying content (e.g., re-partitioning). An implementation supporting dynamic allocation is preferred, but an implementation MAY divide the storage on the device between the two uses in either a fixed or a dynamic manner.

The MediaStorageOption provides an overloaded initialize() method that allows a highly-privileged application to specify the amount of space to be allocated to each use. Implementations SHALL support the reallocation of space between the two uses. Because an OCAP application may change the allocation even on internal storage devices, implementations that cannot dynamically shift storage between the two uses SHOULD store any internal files on a separate reserved section of the device that would not be affected by a reallocation of storage. Implementations that cannot dynamically shift storage between the uses SHOULD NOT initialize a StorageProxy capable of DVR media storage until explicitly requested by an application. Implementations that do initialize such a StorageProxy before explicitly requested SHALL allocate for general-purpose LogicalStorageVolumes at least 3% of the combined storage that can be allocated to both uses. However, implementations are not required to preallocate more than 1GB to general purpose LogicalStorageVolumes.

6.2.1.4.3   Detachable Storage[19]

Storage of recordings to media storage volumes contained within detachable and/or removable storage devices MAY be supported. When supported, the behavior specified in the following sub-sections SHALL apply. For the sake of brevity, where detachable storage is discussed in general, removable storage is also considered included.

In all cases, RecordingChangedEvents SHALL NOT be generated indicating changes that are due to detachment or reattachment of storage. Instead, MediaStorageEvents, which provide access to the list of affected recordings, SHALL be generated.

6.2.1.4.3.1   Scheduling Recordings

The detachment and subsequent reattachment of a detachable storage device SHALL NOT have an effect on the ability of an application to schedule recordings.

---

[19] Added per ECN OCAP-DVR-N-07.1076-2 by kb 11/19/07.

6.2.1.4.3.2   Pending Recordings

The detachment or reattachment of a detachable storage device SHALL NOT have any immediate effect on a pending recording (including the current state), other than the reflection of the readiness of the destination storage via `OcapRecordingRequest.isStorageReady()`.

Implementations SHOULD ensure that sufficient internal storage is available to persistently store scheduled recording information. If sufficient persistent storage is not available for storage of scheduled recordings, recordings MAY be implicitly canceled following a power-cycle.

If, at recording start time, the destination storage is absent, the recording SHALL enter the `IN_PROGRESS_WITH_ERROR_STATE` with a `RecordingFailedException`, where the reason is `RESOURCES_REMOVED`.

6.2.1.4.3.3   In Progress Recordings

The detachment and reattachment of the destination storage for an in-progress recording SHALL be treated as the loss and gain of a resource necessary for recording. The semantics of recording interruption due to resource loss described in [A088 Rev. 2], Section: 6.2.1.2 Recording with resource interruption, SHALL be followed.

6.2.1.4.3.4   Completed Recordings

The detachment of the destination storage for completed recordings SHALL result in the logical deletion of those recordings and removal from the `RecordingManager` database. When a detached storage device is reattached, any recordings that were deleted as a result of the previous detachment that remain on the device SHALL be reconstructed in the `RecordingManager` database.

For detachable devices, such logical deletion occurs as part of making the device detachable.

For removable storage devices, such logical deletion occurs as part of ejecting the storage medium.

6.2.1.5   Lightweight Triggers[20]

The DVB-GEM [DVB-GEM 1.0.2] Annex P Lightweight binding of trigger API is extended to include an API that allows an application to synchronize content with private data, where stream events are signaled in a proprietary fashion. An application that is familiar with proprietary signaling can use the DAVIC MPEG section filtering API to discover it.

An application can register itself as a `LightweightTriggerHandler` (handler) interested in availability of a specific broadcast service stream type (e.g., private data). It does this by calling the `LightweightTriggerManager.registerHandler` method. The implementation will call the `LightweightTriggerHandler.notifyStreamType` method to inform an application when a stream of interest is available for section filtering. The implementation SHALL NOT report a requested stream type for any services other than broadcast services.

The `notifyStreamType` method takes a `LightweightTriggerSession` object that can be used to populate the `DSMCCStreamEvent` with stream events associated with proprietary data filtered from the stream. The session object contains a Locator created by the implementation for the artificial carousel. The implementation SHALL create an artificial carousel for the session. This carousel can be attached to as soon as an application receives a `LightweightTriggerSession` from the `notifyStreamType` method. However, it will not contain any events until added by an application using the `LightweightTriggerSession.registerEvent` method. For service domain attach purposes, the

---

[20] Added per ECN OCAP-DVR-N-06.0911-10 by kb 4/12/07.

carousel SHALL be named "lightweight_triggers" as per DVB-GEM [DVB-GEM 1.0.2] Annex P; see section P.2.3.1.

During an active `LightweightTrigger` session, the `LightweightTriggerSession.store` method MAY be called. This causes the artificial carousel associated with the session to be stored with any permanent recording created for the stream the carousel was created for. It is implementation-specific regarding how the carousel is stored with a permanent recording.

Once an artificial carousel is created, it can be attached to using the DSMCC API. Events that were added to an artificial carousel by a handler application can be listened for during broadcast service time-shift or recorded service playback. For events to be generated with a recording playback, an artificial carousel must be stored with the recording using the `LightweightTriggerSession.store` method. However, for broadcast service time shift, a `LightweightTriggerSession.store` call is not necessary. Events from an artificial carousel SHALL be generated for presenting services only. To attach to an artificial carousel during broadcast service presentation, an application can pass the Locator returned from the `LightweightTriggerSession.getLocator` method to the `ServiceDomain.attach(Locator)` method. An artificial carousel can be attached to for broadcast presentation while the corresponding `LightweightTriggerSession` is open. In addition, it can be attached to after the session is stopped as long as events that were registered for the carousel can still be triggered by trick-mode and normal play of buffered content that was stored for the service the session pertained to. Once content that can be associated with any registered event is no longer available, the artificial carousel for a broadcast service is considered permanently lost unless stored with a recording.

Completed recording playback can be started at arbitrary times, and an application can register itself to listen for this type of playback using the `OcapRecordingManager.addRecordingPlaybackListener`. A `RecordingPlaybackListener` is passed to this method and contains a `notifyRecordingPlayback` method that SHALL be called by the implementation whenever a permanent recording playback is started. Applications MAY use the `ServiceDomain.attach(Locator locator, int carouselId)` method to attach to artificial carousels in permanent recording playback, where the `locator` is the implementation-specific locator of the recorded service and the `carouselId` is the Id used to created the artificial carousel. An artificial carousel stored with a completed recording can be attached to anytime while the recording is being played back in a `ServiceContext` regardless of trick mode. Once the `ServiceContext` is stopped or destroyed, any attached artificial carousel is considered permanently lost.

Certain artificial carousel class and interface methods require special handling as follows:

ServiceDomain

- `getLocator` – Returns an implementation-specific locator.

- `getMountPoint` – Returns a read-only DSMCCObject created by the implementation for the artificial carousel.

- `isNetworkConnectionAvailable` – Returns true as long as any events in the artificial carousel are available during time-shifting or any events are stored with a permanent recording.

DSMCCObject (returned from `ServiceDomain.getMountPoint` for an attached artificial carousel and used in a call to `DSMCCStreamEvent(DSMCCObject)`)

- `getPath` – Returns an implementation-specific value that SHALL be unique within the implementation.

- `list` – Returns an empty array from either list overloaded method.

- `setRetrievalMode` – Because artificial carousels are always in cache, once the carousel is loaded this method does nothing successfully.

- `isStreamEvent` – Always returns true.

DSMCCStream

- `getNPT` – Returns 0 without blocking.

- `isAudio` – Returns false.

- `isData` – Returns true.

- `isMPEGProgram` – Returns false.

- `isVideo` – Returns false.

DSMCCStreamEvent

- `DSMCCStreamEvent(DSMCCObject)` constructor shall throw `NotLoadedException` if either the corresponding `DSMCCObject.asynchronousLoad` or `DSMCCObject.synchronousLoad` methods have not been called for the artificial carousel.

- `subscribe` – The implementation SHALL map the event name used in calls to `LightweightTriggerSession.registerEvent` to this method.

- `unsubscribe` – The implementation SHALL map the event name and Id used in calls to `LightweightTriggerSession.registerEvent` to the respective overloaded method.

- `getEventList` – Returns the set of registered events that are contained within a recording or TSB duration. When an event is only contained within a TSB duration and the time of the event falls outside the duration of the buffer, the event SHALL not be contained in the set of events returned by this method.[21]

StreamEvent

- `getEventNPT` – Returns 0 without blocking.

ObjectChangeEventListener[22]

- `receiveObjectChangeEvent` – The implementation SHALL call this method whenever an event is registered by the `LightweightTriggerSession.registerEvent` method. The implementation SHALL call this method whenever a registered event is buffered in a TSB and the time of the event falls outside the duration of the TSB.

ObjectChangeEvent[23]

- The contained version number SHALL be implementation specific.

When an application attaches to an artificial carousel during completed recording playback, the `DSMCCStreamEvent` SHALL be copied from persistent storage to volatile storage, and the retrieval mode SHALL always be `FROM_CACHE`.

When an application is listening for stream events from an artificial carousel, the implementation SHALL generate them in playback and all trick-mode media rates. The accuracy of the time of application receipt of an application created stream event as compared to the time the event was created with is implementation-specific.

---

[21] Added by ECN OCAP-DVR-N-07.1048-2 by kb 11/19/07.
[22] Added by ECN OCAP-DVR-N-07.1048-2 by kb 11/19/07.
[23] Added by ECN OCAP-DVR-N-07.1048-2 by kb 11/19/07.

7   RECORDING AND PLAYBACK APIS

This section describes recording and playback APIs particular to the OCAP DVR platform.

7.1   DVB-GEM Specification Correspondence

Section 7 Recording and Playback API (this section) corresponds to [A088 Rev. 2], Chapter 7 as follows:[24]

Table 7–1 - Correlation between OCAP 1.0 and [A088 Rev. 2] MHP PVR/PDR Common Core Specification

OCAP Compliance	MHP PVR/PDR Common Core Specification Section	GEM Compliance
7.2.1.2, Recording API	7 Recording and playback APIs	Extension
7.2.1.1, Recording and Power Modes	7.1 Recording and recording management	Extension
No corresponding section	7.2 Playback	Complete compliance
7.2.1.2, Recording API	No corresponding section	Extension
7.2.1.3, OCAP DVR API	No corresponding section	Extension
No corresponding section	7.3 Other APIs	Complete compliance
7.2.1.4, Permissions	7.4 Recording and recording management	Extension
No corresponding section	7.4.1 Unsigned applications	Complete compliance
7.2.1.4.1, Signed Applications	7.4.2 Signed applications	Extension
7.2.1.4.2, Monitor Application Permission	No corresponding section	Extension

7.2   OCAP 1.0 DVR Specific Requirements

7.2.1   Extensions to [A088 Rev. 2] MHP PVR/PDR Common Core

7.2.1.1   Recording and Power Modes[25]

The OCAP DVR specification extends [A088 Rev. 2], Section 7.2.1.1.

An initial monitor application can delay scheduled recording start by calling the `org.ocap.dvr.OcapRecordingManager.setRecordingDelay()` method. When this method is called, the implementation SHALL delay the start of scheduled recordings by the amount of time passed in after the time the initial monitor application calls the `org.ocap.OcapSystem.monitorConfiguredSignal` method or the timeout to call that method expires. If the `setRecordingDelay` method is called after the `monitorConfiguredSignal` method is called or times out, the implementation SHALL NOT apply a delay to scheduled recordings start. The `org.ocap.dvr.OcapRecordingManager.signalRecordingStart` method can be used to terminate the delay timeout if it is in effect.

---

[24] Revised Table 7–1 per ECN OCAP-DVR-N-05.0805-5 by GO on 12/20/05.

[25] Added this subsection per ECN OCAP-DVR-N-05.0805-5 by GO on 12/20/05; Revised this subsection per ECN OCAP-DVR-N-05.0856-2 and OCAP-DVR-N-06.0887-2 by GO on 9/25/06.

When the implementation is in low-power mode and a scheduled recording is about to start, the implementation SHALL power-on any devices required to complete the recording, e.g., hard-drive. The implementation SHALL power-on needed devices ahead of time so that the needed devices are ready when the recording begins. If powering up such devices violates low-power mode power consumption requirements, the implementation MAY take the Host device to full-power. In this case, output ports and integrated displays disabled for low power mode SHALL remain disabled. In addition, the `org.ocap.hardware.Host.getPowerMode` SHALL return `LOW_POWER`, and the implementation SHALL NOT generate an event for power mode change. The duration required to prepare devices for usage is implementation-specific.

7.2.1.2   Recording API

The OCAP DVR platform extends [A088 Rev. 2] and adds support for the following packages:

```
org.ocap.dvr
org.ocap.dvr.storage
org.ocap.dvr.event
```

7.2.1.3   OCAP DVR API[26]

The OCAP DVR platform extends the OCAP-J API defined in [OCAP1.0]. The additional packages, classes, and interfaces are listed here. For a complete definition and description of the APIs, see Annex D, OCAP DVR API (org.ocap.dvr); Annex E, OCAP DVR Storage API (org.ocap.dvr.storage); OCAP Shared DVR API (org.ocap.shared.dvr) - see [A088 Rev. 2]; OCAP Shared DVR Navigation API (org.ocap.shared.dvr.navigation) - see [A088 Rev. 2]; OCAP Shared Media API (org.ocap.shared.media) [A088 Rev. 2]; Annex J, OCAP DVR Event API (org.ocap.dvr.event). An implementation of the OCAP DVR platform SHALL include all of the packages, classes, and interfaces defined in [OCAP1.0] and [A088 Rev. 2], as well as the packages, classes, and interfaces required by this section. An additional class, org.davic.media.MediaTimeEventControl, is required by [A088 Rev. 2]. These are enhancements to the APIs defined in [OCAP1.0], as well as the following packages:

OCAP DVR Extensions to GEM

```
org.ocap.dvr.storage.MediaStorageOption
```

```
org.ocap.dvr.storage.MediaStorageVolume
```

```
org.ocap.dvr.storage.SpaceAllocationHandler
```

```
org.ocap.dvr.storage.FreeSpaceListener
```

```
org.ocap.dvr.event.LightweightTriggerHandler
```

```
org.ocap.dvr.event.LightweightTriggerManager
```

```
org.ocap.dvr.event.LightweightTriggerSession
```

```
org.ocap.dvr.event.StreamChangeListener
```

26Revised this subsection per ECN OCAP-DVR-N-05.0805-5 and OCAP-DVR-N-06.0817-8 by GO on 12/20/05 and 9/25/06.

7.2.1.4　Permissions

7.2.1.4.1　Signed Applications

No additional `RecordingPermission` is assigned to signed applications. `RecordingPermissions` with names "create", "modify", "delete" or "cancel" SHALL NOT be granted to a signed application unless the signed application has `MonitorAppPermission("recording")` or `MonitorAppPermission("handler.recording")`.

7.2.1.4.2　Monitor Application Permission[27]

Host devices that implement the OCAP DVR Extension SHALL support the `MonitorAppPermission("recording")` and `MonitorAppPermission("handler.recording")` permission names. Annex Q of [OCAP1.0] is extended as follows:

The table within the description of MonitorAppPermission is extended to include the following rows:

Permission Name	What the Permission Allows	Description
Recording	Allows management of system-wide recording operations	Applications with this permission can delay the start of scheduled recordings and disable system-wide buffering.
Handler.recording	Allows management of recording prioritization and resolution	Applications with this permission can register as request resolution handler, register as space allocation handler and manipulate prioritization of scheduled recordings.

In addition, Section 14.2.2.1.1 in [OCAP1.0] is extended as follows: the enumerated token value type of the `name` attribute of the `ocap:monitorapplication` element type defined by the DTD of the PRF SHALL be considered to contain the "recording" and "handler.recording" values.[28]

Applications with the `MonitorAppPermission("recording")` permission can delay the start of scheduled recordings using the `OcapRecordingManager.setRecordingDelay()` method. Additionally, applications with this permission can control system wide buffering using the `OcapRecordingManager.enableBuffering()` and `OcapRecordingManager.disableBuffering()` methods.

Applications with the `MonitorAppPermission("recording.handler")` permission can register as the space allocation handler and request resolution handler using the `OcapRecordingManager.setSpaceAllocationHandler()` and `OcapRecordingManager.setRequestResolutionHandler()` methods, respectively. Additionally, applications with this permission can modify recording prioritization using the `OcapRecordingManager.getPrioritizedResourceUsages()` and `OcapRecordingManager.setPrioritization()` methods.

An application with `MonitorAppPermission("recording")` is assigned the following set of permissions defined in `org.ocap.shared.dvr.RecordingPermission`:

　　a) `RecordingPermission("create", "own")` - schedule a `RecordingRequest`.

　　b) `RecordingPermission("read", "own")` - obtain the list of `RecordingRequests`.

---

[27] Section modified extensively per ECN OCAP-DVR-N-07.0994-2 by kb 4/13/07.
[28] Modified by ECN OCAP-DVR-N-07.1062-1 by kb 11/19/07.

   c) `RecordingPermission("modify", "own")` - modify properties or application-specific data for a `RecordingRequest`.

   d) `RecordingPermission("delete", "own")` - delete a `RecordingRequest` including recorded content.

   e) `RecordingPermission("cancel", "own")` - cancel a pending `RecordingRequest`.

An application with `MonitorAppPermission("handler.recording")` is assigned the following set of permissions defined in `org.ocap.shared.dvr.RecordingPermission`:

   `RecordingPermission("*", "*")` - create, read, modify, delete or cancel any `RecordingRequest` or `RecordedService`, regardless of any restrictions specified through the extended file access permission associated with the `RecordingRequest`.

8 SIGNALING

This chapter is in complete compliance with [A088 Rev. 2], Chapter 8.

9   APPLICATION MODEL

This chapter is in complete compliance with [A088 Rev. 2], Chapter 9.

10 SECURITY

This section describes security features particular to the OCAP DVR platform. These are enhancements to the security features defined in [OCAP1.0].

10.1 DVB-GEM Specification Correspondence

Section 10 Security (this section) corresponds to [A088 Rev. 2], Chapter 10 as follows:

Table 10–1 - Correlation between OCAP 1.0 and [A088 Rev. 2] MHP PVR/PDR Common Core Specification

OCAP Compliance	MHP PVR/PDR Common Core Specification Section	GEM Compliance
10, Security	10 Security	Complete compliance
10.2.1.1, Access Scope of Recordings	No corresponding section	Extension
10.2.1.2, Content Protection	No corresponding section	Extension
10.2.1.3, Minimum Security Constraints	No corresponding section	Extension

10.2 OCAP 1.0 DVR Specific Requirements

10.2.1 Extensions to [A088 Rev. 2] MHP PVR/PDR Common Core

10.2.1.1 Access Scope of Recordings

Applications MAY indicate an access scope at the time a recording is added to the recording database. An application MAY allow any other application to play back a recording it initiates; it MAY restrict playback to applications from a specific set of organizations; it MAY restrict playback to applications that are from the same organization; or, it MAY restrict playback to itself. The implementation SHALL respect scope parameters and not decrypt, decode, and display recordings that are not within the defined scope (see `org.ocap.storage.ExtendedFileAccessPermission` and the overloaded `org.ocap.dvr.RecordingManager.record()` method).

The following table describes the security restrictions for individual recording requests. Where calling applications do not have extended file access permission as listed in the table below, the corresponding method SHALL throw AccessDeniedException; see the java doc for each respective method: [29]

Table 10–2 - Security restrictions for individual recording requests

Method	Policy
`RecordingManager.addRecordingChanged Listener(RecordingChangedListener)`	No additional permissions required.
`RecordingManager.getEntries()`	Only entries for which the application has read extended file access permission will be listed.
`RecordingRequest.removeAppData(int)`	Permitted only for entries for which the application has write extended file access permission.

---

[29]Revised this paragraph and Table 10-2 per ECN OCAP-DVR-N-05.0836-3 by GO on 1/31/06.

Method	Policy
`RecordingRequest.setRecordingProperties (RecordingSpec)`	Permitted only for entries for which the application has write extended file access permission.
`RecordingRequest.addAppData (String,Serializable)`	Permitted only for entries for which the application has write extended file access permission.
`RecordingRequest.removeAppData(String)`	Permitted only for entries for which the application has write extended file access permission.
`RecordingRequest.delete()`	Permitted only for entries for which the application has write extended file access permission.
`RecordingManager.record()`	No additional permissions required.
`ParentRecordingRequest.cancel()`	Permitted only for entries for which the application has write extended file access permission.
`RecordedService.setMediaTime(Time)`	Permitted only for entries for which the application has write extended file access permission.
`RecordedService.delete()`	Permitted only for entries for which the application has write extended file access permission.
`LeafRecordingRequest.getService()`	No additional permissions required
`LeafRecordingRequest.stop()`	Permitted only for entries for which the application has write extended file access permission.
`LeafRecordingRequest.cancel()`	Permitted only for entries for which the application has write extended file access permission.

10.2.1.2 Content Protection

The OCAP DVR platform represents an extension of the cable network operator's service. As such, the platform respects the copy-protection rules defined in [OCAP1.0]. Because recorded content originates from the cable network, all content SHALL be recorded in a manner that unites the content to the network on which it was recorded. Any network-specific information saved by the Host device for this purpose must either be encrypted, or placed in a secure storage device location that cannot be read from outside the device. Furthermore, in order to meet copy-control requirements, all content SHALL be encrypted in a manner that unites the content to the device which initiated its recording. These features are, in general, transparent to a viewer, as long as the recording device is not connected to a different network, or a storage device is not connected to a different receiver.

The means by which a recording is associated with the network, and the device from which it was recorded, will be defined in a future revision of this Profile.

10.2.1.3 Minimum Security Constraints

Minimum "level of security" requirements are asserted by this Profile as follows:

- For DES implementations - The Triple DES Encryption Algorithm (TDEA) as per [FIPS-46-3] SHALL be the minimum DES configuration.

- For AES implementations - The minimum configuration SHALL be AES-128 as defined by [FIPS-197].

This Profile complies with the OpenCable System Security Specification [OC-SEC] regarding [FIPS-140-2] Level 1 security requirements and extends those requirements to any recording components.

11 MINIMUM PLATFORM CAPABILITIES

This chapter describes the minimum platform capabilities of the OCAP DVR platform.

11.1 DVB-GEM Specification Correspondence

Section 11, Minimum Platform Capabilities (this section) does not correspond to any [A088 Rev. 2] chapter, as depicted below:

Table 11–1 - Correlation between OCAP 1.0 and [A088 Rev. 2] MHP PVR/PDR Common Core Specification

OCAP Compliance	MHP PVR/PDR Common Core Specification Section	GEM Compliance
11, Minimum Platform Capabilities	No corresponding section	Extension
11.2.1.1, Bit Rate	No corresponding section	Extension
11.2.1.2, Storage Devices	No corresponding section	Extension
11.2.1.3, Time-shift Buffers	No corresponding section	Extension
11.2.1.5, OpenCable Set-top Terminal Core Requirements	No corresponding section	Extension

11.2 OCAP 1.0 DVR Specific Requirements

11.2.1 Extensions to A088 MHP PVR/PDR Common Core

11.2.1.1 Bit Rate

Implementations SHALL support three recording bit-rates: low, medium, and high. Where high causes a recording to be encoded with the best audio/video quality, but takes more storage space than the other settings. For analog recordings. these values are implementation-specific. Bit-rates for digital recordings are related to transrating, which is a process of modifying MPEG compression to achieve greater compression. Transrating is optional, but when supported, high bit-rate is equivalent to no transrating, and medium to low bit rates specify increasing compression. When transrating is supported, the type of transrating supported is implementation-specific.

11.2.1.2 Storage Devices

Implementations SHALL enable simultaneous recording and playback to/from given storage devices, when supported by corresponding storage devices. At least one storage device provided by the platform SHALL support simultaneous recording and playback. Implementations SHALL enable recording to one device, and playback from any other, if multiple storage devices are present.

11.2.1.3 Time-shift Buffers

Implementations SHALL support at least one time-shift buffer. If a Host device supports simultaneous presentation of services, the implementation MAY support multiple time-shift buffers. If multiple time-shift buffers are supported, the implementation MAY support allocation of time-shift buffers and attachment of timeshift buffers to various service contexts.

11.2.1.4 Recording Multiple Streams from the Same Service

Implementations SHALL support simultaneous recording of at least one video stream and at least two audio streams that are broadcast as a part of a broadcast service. If a broadcast service contains more than one video stream, implementations SHALL record at least the first video stream in the PMT elementary stream loop. If the broadcast service contains more than two audio streams, the implementation shall record the audio stream matching the user preferred language, if available. If no match is found, the implementation SHALL record the first audio stream in the PMT elementary stream loop, as the preferred audio. If more than one audio stream is available in the broadcast service, the implementation SHALL arbitrarily pick a second audio stream of a different language from the first to store with the recorded service. It is implementation-dependent regarding whether additional audio or video streams available in the service being recorded are actually stored with the recorded service.

11.2.1.5 OpenCable Set-top Terminal Core Requirements

The resources as specified for all OCAP profiles in the [HOST2.0] are REQUIRED. Implementations SHALL also comply with the requirements of [HOST2-DVR].

Annex A   Application Recording Description (Normative)

This annex defines requirements for application recording.

A.1   DVB-GEM Specification Correspondence

This Annex corresponds to [A088 Rev. 2], Annex A, as follows:

Table A–1 - Correlation between OCAP 1.0 and [A088 Rev. 2] MHP PVR/PDR Common Core Specification

OCAP Compliance	MHP PVR/PDR Common Core Specification Section	GEM Compliance
Annex A, Application Recording Description (Normative)	Annex A (normative) Application recording description	Complete compliance

Annex B   Responsibilities of this Specification (Informative)

This annex complies with [A088 Rev. 2], Annex B (informative), Responsibilities of GEM Recording Specifications. The responsibilities called out in the referenced annex are defined in the following sections of this specification.

Table B–1 - Mapping for GEM Required Responsibilities

	GEM Requirement	OCAP DVR Spec Section
1	Which types of streams are to be considered as "recordable streams". Stream types corresponding to clauses 7.2.1 ("Audio") and 7.2.2 ("Video") of GEM in the GEM terminal specification on which the GEM recording specification is based, must be considered as recordable streams.	Section 6.2.1.1.3.1, Identifying Streams to be Recorded
2	Mechanisms for resolving conflicts between requested recordings (e.g., use of the tuner).	Section 6.2.1.1.6, Resource Management for Recording Requests
3	Minimum capabilities for the number of steams (or number of streams of each type) that a GEM recording terminal must be able to record.	Section 11.2.1.4, Recording Multiple Streams from the Same Service
4	The definition of which applications are recordable in both scheduled and time-shift recording (which need not be the same).	Section 6.2.1.1.3.2, Identifying and Recording Applications and Section 6.2.1.3.2.2, Identifying and Recording Applications
5	Requirements on a GEM recording terminal to monitor for dynamic data (in the DSMCC object carousel or GEM functional equivalent), during scheduled and time-shift recording (which need not be the same).	Section 6.2.1.1.3.2, Identifying and Recording Applications and Section 6.2.1.3.2.2, Identifying and Recording Applications
6	Requirements on a GEM recording terminal to monitor for GEM triggers or DSMCC stream events during scheduled and time-shift recording (which need not be the same).	Section 6.2.1.1.3.2, Identifying and Recording Applications and Section 6.2.1.3.2.2, Identifying and Recording Applications
7	Requirements on a GEM recording terminal to monitor for dynamic application signaling during scheduled and time-shift recording (which need not be the same).	Section 6.2.1.1.3.2, Identifying and Recording Applications and Section 6.2.1.3.2.2, Identifying and Recording Applications
8	Requirements on reconstructing the dynamic behavior of recorded applications during playback of scheduled and time-shift recordings (which need not be the same).	Section 6.2.1.1.3.2, Identifying and Recording Applications and Section 6.2.1.3.2.2, Identifying and Recording Applications
9	How accurately the expiration period should be enforced by implementations.	Section 6.2.1.1.4, Managing Completed Recordings
10	The definition of at least one protocol for transmitted time lines.	Section 6.2.1.1.3.1, Identifying Streams to be Recorded
11	The conditions when a JMF player or service context has a time-shift buffer attached.	Section 6.2.1.3. Time-Shift Buffer
12	A mechanism to associate security attributes with individual recording requests. A mapping from that mechanism to the language in each of the methods in Table 10–2.	Section 10.2.1.1, Access Scope of Recordings Table 10–2 - Security restrictions for individual recording requests
13	Requirements on the number of entries of application-specific private data that it must be possible to associate with a single RecordingRequest without a NoMoreDataEntriesException being thrown.	Defined as a minimum of 16 entries in org.ocap.shared.dvr.RecordingRequest.addApp Data [A088 Rev. 2].

	GEM Requirement	OCAP DVR Spec Section
14	A mechanism to associate security attributes with individual recording requests and a mapping from that mechanism to the language in each of the methods in Table 10–2.	Section 10.2.1.1, Access Scope of Recordings Table 10–2 - Security restrictions for individual recording requests
15	The mechanism for resolving parent recording requests, including setting the initial state of a parent recording request.	6.2.1.1.7, Request Resolution Process
16	The events generated during playback when the start and end of a recording are reached.	6.2.1.2.1, Process for Playback10F

Annex C   External References; Errata, Clarifications, and Exemptions (Normative)

This annex defines clarifications for JMF usage.

C.1   DVB-GEM Specification Correspondence

This Annex corresponds to [A088 Rev. 2], Annex C, as follows:

Table C–1 - Correlation between OCAP 1.0 and [A088 Rev. 2] MHP PVR/PDR Common Core Specification

OCAP Compliance	MHP PVR/PDR Common Core Specification Section	GEM Compliance
Annex C External References; Errata, Clarifications, and Exemptions (Normative)	Annex C (normative) External references; errata, clarifications, and exemptions	Complete compliance

Annex D   OCAP DVR API (org.ocap.dvr)

Package org.ocap.dvr

OCAP Specific extensions to the shared DVR API.

Interface Summary	
OcapRecordedService	This interface represents a RecordedService in OCAP.
OcapRecordingRequest	This interface represents a LeafRecordingRequest in OCAP.
RecordingAlertListener	Listener for Recording Alerts.
RecordingPlaybackListener	This interface represents a listener that can be added to listen for recording playback start.
RecordingResourceUsage	This interface represents a grouping of resources specific to a recording function performed by an application.
RequestResolutionHandler	This interface will be implemented by the application that registers the RequestResolutionHandler.
SharedResourceUsage	This interface represents a group of resources where one or more resources are shared between multiple resource usages.
TimeShiftBufferResourceUsage	This interface represents a grouping of resources specific to a time-shift buffering performed by an application.
TimeShiftListener	The TimeShiftListener interface is implemented by applications wishing to receive events related to time shift resources.
TimeShiftProperties	This interface represents a set of time-shift properties that can be set for and queried from a `ServiceContext`.

Class Summary	
BufferingRequest	This class represents an application request for buffering.
OcapRecordingManager	RecordingManager represents the entity that performs recordings.
OcapRecordingProperties	Encapsulates the details about how a recording is to be made.
PrivateRecordingSpec	Specifies a recording request that can be resolved only by an application defined request resolution handler.
RecordingAlertEvent	Event notifying that a scheduled recording is about to occur.
TimeShiftEvent	The parent class for TimeShiftBuffer events.

org.ocap.dvr
Class BufferingRequest

```
java.lang.Object
  └─org.ocap.dvr.BufferingRequest
```

```
public abstract class BufferingRequest
extends java.lang.Object
```

This class represents an application request for buffering. An application can call the createInstance method to create a request.

Constructor Summary

protected	**BufferingRequest**() Protected constructor, not to be used by applications.

Method Summary

static BufferingRequest	**createInstance** (javax.tv.service.Service service, long minDuration, long maxDuration, org.ocap.storage. ExtendedFileAccessPermissions efap) Creates a BufferingRequest object.
abstract org.dvb.application.AppID	**getAppID**() Gets the AppID of the application that created the request.
abstract org.ocap.storage. ExtendedFileAccessPermissions	**getExtendedFileAccessPermissions**() Gets the ExtendedFileAccessPermissions for this request.
abstract long	**getMaxDuration**() Gets the maximum duration to buffer for this request.
abstract long	**getMinimumDuration**() Gets the minimum content buffering duration for this request.
abstract javax.tv.service.Service	**getService**() Gets the Service this request is attempting to buffer.
abstract void	**setExtendedFileAccessPermissions** (org.ocap.storage. ExtendedFileAccessPermissions efap) Sets the ExtendedFileAccessPermissions for this request.
abstract void	**setMaxDuration**(long duration) Sets the maximum duration of content that MAY be buffered for this BufferingRequest.
abstract void	**setMinimumDuration**(long minDuration) Sets the minimum duration of content that SHALL be buffered for this request.
abstract void	**setService**(javax.tv.service.Service service) Sets the Service this request is attempting to buffer.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

BufferingRequest

protected **BufferingRequest**()

>Protected constructor, not to be used by applications.

Method Detail

createInstance

public static BufferingRequest
createInstance(javax.tv.service.Service service,
                                                 long minDuration,
                                                 long maxDuration,

org.ocap.storage.ExtendedFileAccessPermissions efap)

>Creates a BufferingRequest object.
>**Parameters:**
>service - The service to buffer.
>minDuration - Minimum duration in seconds to buffer.
>maxDuration - Maximum duration in seconds to buffer.
>efap - Extended file access permissions for this request. If this parameter is null, no write permissions are given to this request. Read permissions for BufferingRequest instances are always world regardless of read permissions set by this parameter.
>**Throws:**
>java.lang.IllegalArgumentException - if the service parameter is not a valid Service, or if minDuration is less than OcapRecordingManager.getSmallestTimeShiftDuration(), or if maxDuration is less than minDuration.

getService

public abstract javax.tv.service.Service **getService**()

>Gets the Service this request is attempting to buffer.
>**Returns:**
>Service being buffered for this request.

setService

public abstract void **setService**(javax.tv.service.Service service)
                            throws java.lang.IllegalArgumentException

>Sets the Service this request is attempting to buffer.
>**Parameters:**
>service - The Service to buffer for this request.
>**Throws:**
>java.lang.IllegalArgumentException - if the parameter is not a valid Service.

java.lang.SecurityException - if the calling applications does not have one of the write ExtendedFileAccessPermissions set by the createInstance or setExtendedFileAccessPermissions methods.

getMinimumDuration

public abstract long **getMinimumDuration**()

Gets the minimum content buffering duration for this request.

Returns:

The minimum content buffering duration in seconds.

setMinimumDuration

public abstract void **setMinimumDuration**(long minDuration)

Sets the minimum duration of content that SHALL be buffered for this request. If this method necessitates a buffer re-size the implementation MAY flush the contents of the buffer.

Parameters:

minDuration - Minimum duration in seconds.

Throws:

java.lang.IllegalArgumentException - If the parameter is greater than the current value and Host device does not have enough space to meet the request, or if the parameter is greater than the maximum duration set by the createInstance or setMaximumDuration methods, or if the parameter is less than the duration returned by OcapRecordingManager.getSmallestTimeShiftDuration().

java.lang.SecurityException - if the calling application does not have one of the write ExtendedFileAccessPermissions set by the createInstance or setExtendedFileAccessPermissions methods.

getMaxDuration

public abstract long **getMaxDuration**()

Gets the maximum duration to buffer for this request. Returns the value set by the createInstance or setMaximumDuration methods.

Returns:

Maximum duration in seconds.

setMaxDuration

public abstract void **setMaxDuration**(long duration)

Sets the maximum duration of content that MAY be buffered for this BufferingRequest. Informs the implementation that storing more content than this is not needed by the application owning this BufferingRequest.

Parameters:

duration - The maximum duration in seconds.

Throws:

java.lang.IllegalArgumentException - if the duration parameter is negative or if the parameter is less than the minimum duration set by the createInstance or setMaximumDuration methods, or if the parameter is less than the duration returned by OcapRecordingManager.getSmallestTimeShiftDuration().

java.lang.SecurityException - if the calling application does not have one of the write ExtendedFileAccessPermissions set by the createInstance or setExtendedFileAccessPermissions methods.

---

getExtendedFileAccessPermissions

```
public abstract org.ocap.storage.ExtendedFileAccessPermissions
getExtendedFileAccessPermissions()
```
> Gets the ExtendedFileAccessPermissions for this request.
> **Returns:**
> The ExtendedFileAccessPermissions.

setExtendedFileAccessPermissions

```
public abstract void
setExtendedFileAccessPermissions(org.ocap.storage.ExtendedFileAccessPermission
s efap)
```
> Sets the ExtendedFileAccessPermissions for this request.
> **Parameters:**
> `efap` - The ExtendedFileAccessPermissions for this request.
> **Throws:**
> `java.lang.IllegalArgumentException` - if the parameter is null;
> `java.lang.SecurityException` - if the calling application is not the creator of this request.

getAppID

```
public abstract org.dvb.application.AppID getAppID()
```
> Gets the AppID of the application that created the request. If null is returned the implementation created
> the request.
> **Returns:**
> AppID of the owning application.

org.ocap.dvr
Interface OcapRecordedService

All Superinterfaces:
>   org.ocap.shared.dvr.RecordedService, javax.tv.service.Service

```
public interface OcapRecordedService
extends org.ocap.shared.dvr.RecordedService
```

This interface represents a RecordedService in OCAP. The object returned when an applications calls the getService method on a RecordingRequest will be an instance of this interface.

Method Summary

long	**getRecordedBitRate**() Get the bit-rate used for encoding and storage of this recorded service.
long	**getRecordedSize**() Gets the size of the recording in bytes.
boolean	**isDecodable**() Determines if the recording has a format which can be decoded for presentation by the implementation, e.g., the bit rate, resolution, and encoding are supported.
boolean	**isDecryptable**() Determines if the recording can be decrypted by the implementation on the current network.

Methods inherited from interface org.ocap.shared.dvr.RecordedService

```
delete, getFirstMediaTime, getMediaLocator, getMediaTime, getRecordedDuration,
getRecordingRequest, getRecordingStartTime, setMediaTime
```

Methods inherited from interface javax.tv.service.Service

```
equals, getLocator, getName, getServiceType, hashCode, hasMultipleInstances,
retrieveDetails
```

Method Detail

getRecordedBitRate

```
long getRecordedBitRate()
```
>   Get the bit-rate used for encoding and storage of this recorded service.
>   **Returns:**
>   Bit-rate in bytes per second.

getRecordedSize

```
long getRecordedSize()
```
>   Gets the size of the recording in bytes.
>   **Returns:**
>   Space occupied by the recording in bytes.

isDecryptable

boolean **isDecryptable**()

Determines if the recording can be decrypted by the implementation on the current network.
Returns:
True if the recording can be decrypted, otherwise returns false.

isDecodable

boolean **isDecodable**()

Determines if the recording has a format which can be decoded for presentation by the implementation, e.g. the bit rate, resolution, and encoding are supported.
Returns:
True if the recording can be decoded, otherwise returns false.

org.ocap.dvr
Class OcapRecordingManager

```
java.lang.Object
  └─ org.ocap.shared.dvr.RecordingManager
       └─ org.ocap.dvr.OcapRecordingManager
```

public abstract class **OcapRecordingManager**
extends org.ocap.shared.dvr.RecordingManager

RecordingManager represents the entity that performs recordings. An instance of this class is returned when an application calls the RecordingManager.getInstance() class in OCAP platforms.

The setPrioritization method is intended to be used with the current set of prioritized resource usages that would be returned by a call to the getPrioritizedUsages method. However, the set of resource usages returned by a call to the getPrioritizedResourceUsages MAY be invalid in a subsequent call to the setPrioritization method as the set MAY have changed during the time between the two method calls.

Constructor Summary

OcapRecordingManager()

Method Summary

abstract void	**addRecordingAlertListener**(RecordingAlertListener ral) Adds an event listener for receiving events corresponding to a transition from a pending state to an in-progress state or a failed state.
abstract void	**addRecordingAlertListener**(RecordingAlertListener ral, long alertBefore) Adds an event listener for receiving events corresponding to a transition from a pending state to an in-progress state or a failed state.
abstract void	**addRecordingPlaybackListener** (RecordingPlaybackListener listener) Adds an event listener for receiving events corresponding to a recording playback start.
abstract void	**cancelBufferingRequest**(BufferingRequest request) Cancels an active buffering request.
abstract void	**disableBuffering**() Disables time-shift buffering and buffering without presentation.
abstract void	**enableBuffering**() Enables time-shift buffering and buffering without presentation.
abstract BufferingRequest[]	**getBufferingRequests**() Gets a set of buffering requests that were passed to the requestBuffering method and have not been cancelled.
abstract long	**getMaxBitRate**() Gets the maximum bit rate the implementation will use for duration to space in calculations.

Method Summary

abstract org.ocap.resource. ResourceUsage[]	**getPrioritizedResourceUsages** (org.ocap.shared.dvr.RecordingRequest recording) Get the prioritized list of overlapping ResourceUsages corresponding to a particular recording request.
abstract long	**getSmallestTimeShiftDuration**() Gets the smallest time-shift duration supported by the implementation.
abstract org.ocap.shared.dvr. RecordingRequest	**record**(org.ocap.shared.dvr.RecordingSpec source, java.lang.String[] keys, java.io.Serializable[] appData) Records the stream or streams according to the source parameter.
abstract void	**removeRecordingAlertListener** (RecordingAlertListener ral) Removes a registered event listener for receiving recording events.
abstract void	**removeRecordingPlaybackListener** (RecordingPlaybackListener listener) Removes a registered event listener for receiving recording playback events.
abstract void	**requestBuffering**(BufferingRequest request) Requests the implementation to start buffering a service using implementation specific time-shift storage.
abstract org.ocap.shared.dvr. RecordingRequest	**resolve**(org.ocap.shared.dvr.RecordingRequest request, org.ocap.shared.dvr.RecordingSpec spec, int resolutionState) Schedule a child recording request corresponding to an unresolved or partially resolved recording request.
abstract void	**setPrioritization** (org.ocap.resource.ResourceUsage[] resourceUsageList) Sets the relative priorities for a set of ResourceUsages.
abstract void	**setRecordingDelay**(long seconds) Sets the amount of time to delay the start of scheduled recordings after the initial monitor application is running.
abstract void	**setRequestResolutionHandler** (RequestResolutionHandler rrh) Set the RequestResolutionHandler that will be invoked when any application calls the RecordingManager.record method.
abstract void	**setSpaceAllocationHandler**(SpaceAllocationHandler sah) Set the SpaceAllocationHandler that will be invoked when any application attempts to allocate space in any MediaStorageVolume.
abstract void	**signalRecordingStart**() Informs the implementation it SHALL start scheduled recordings if it hasn't already done so.

Methods inherited from class org.ocap.shared.dvr.RecordingManager
addRecordingChangedListener, getEntries, getEntries, getInstance, getRecordingRequest, record, removeRecordingChangedListener

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,
wait, wait, wait
```

Constructor Detail

OcapRecordingManager

```
public OcapRecordingManager()
```

Method Detail

addRecordingAlertListener

```
public abstract void addRecordingAlertListener(RecordingAlertListener ral)
```
> Adds an event listener for receiving events corresponding to a transition from a pending state to an in-progress state or a failed state. The listener parameter will only be informed of these events for entries the calling application has read file access permission to.
> **Parameters:**
> ral - The listener to be registered.

addRecordingAlertListener

```
public abstract void addRecordingAlertListener(RecordingAlertListener ral,
                                               long alertBefore)
```
> Adds an event listener for receiving events corresponding to a transition from a pending state to an in-progress state or a failed state. The listener parameter will only be informed of these events for entries the calling application has read file access permission to.
> **Parameters:**
> ral - The listener to be registered.
> alertBefore - Time in milliseconds for the alert to be generated before the start of the scheduled event.

removeRecordingAlertListener

```
public abstract void removeRecordingAlertListener(RecordingAlertListener ral)
```
> Removes a registered event listener for receiving recording events. If the listener specified is not registered then this method has no effect.
> **Parameters:**
> ral - the listener to be removed.

addRecordingPlaybackListener

```
public abstract void
addRecordingPlaybackListener(RecordingPlaybackListener listener)
```
> Adds an event listener for receiving events corresponding to a recording playback start.
> **Parameters:**
> listener - The listener to add.

removeRecordingPlaybackListener

```
public abstract void
removeRecordingPlaybackListener(RecordingPlaybackListener listener)
```

Removes a registered event listener for receiving recording playback events. If the listener specified is not registered then this method has no effect.
Parameters:
listener - The listener to be removed.

setSpaceAllocationHandler

public abstract void **setSpaceAllocationHandler**(SpaceAllocationHandler sah)

Set the SpaceAllocationHandler that will be invoked when any application attempts to allocate space in any MediaStorageVolume. At most one instance of this handler can be set. Subsequent calls to this method replace the previous instance with the new one.
Parameters:
sah - the space reservation handler.
Throws:
java.lang.SecurityException - if the caller does not have MonitorAppPermission("handler.recording").

setRequestResolutionHandler

public abstract void **setRequestResolutionHandler**(RequestResolutionHandler rrh)

Set the RequestResolutionHandler that will be invoked when any application calls the RecordingManager.record method. At most only one instance of this handler can be set. Subsequent calls to this method replaces the previous instance with the new one.
Parameters:
rrh - the request resolution handler.
Throws:
java.lang.SecurityException - if the caller does not have MonitorAppPermission("handler.recording").

resolve

public abstract org.ocap.shared.dvr.RecordingRequest
resolve(org.ocap.shared.dvr.RecordingRequest request,

org.ocap.shared.dvr.RecordingSpec spec,

int resolutionState)

Schedule a child recording request corresponding to an unresolved or partially resolved recording request. This method is called either by the RequestResolutionHandler or by an application that has enough information to provide request resolutions. The implementation SHALL generate a recording request corresponding to each successful invocation of this method and make that recording request a child of the RecordingRequest passed in as the first parameter. If the implementation has enough information to resolve the newly created recording request, the implementation should resolve the recording request.

Implementation should set the state of the recording request "request" to "resolutionState" before the return of this call.

Parameters:
request - the RecordingRequest for which the resolution is provided.
spec - the RecordingSpec for the child recording request.
resolutionState - the state of the RecordingRequest after the return of this method. The possible values for this parameter are the states defined in ParentRecordingRequest.
Returns:
the newly scheduled recording request.
Throws:

```
java.lang.SecurityException - if the caller does not have
```
MonitorAppPermission("handler.recording").
```
java.lang.IllegalArgumentException - if the resolutionState is not a state defined in
```
ParentRecordingRequest, or if the request is not in unresolved or partially resolved state.

getPrioritizedResourceUsages

```
public abstract org.ocap.resource.ResourceUsage[]
getPrioritizedResourceUsages(org.ocap.shared.dvr.RecordingRequest recording)
```
Get the prioritized list of overlapping ResourceUsages corresponding to a particular recording request. The list of resource usages may include RecordingResourceUsages and other types of ResourceUsages. The ResourceUsage corresponding to the specified recording request is also included in the prioritized list. The prioritized list is sorted in descending order of prioritization. The prioritization for resource usages is based on OCAP resource management.

Parameters:
`recording` - the RecordingRequest for which overlapping resource usages are sought.

Returns:
the list of ResourceUsages overlapping with the specified RecordingRequest, including the ResourceUsage corresponding to the specified RecordingRequest, sorted in descending order of prioritization, null if the RecordingRequest is not in one of the pending or in-progress states.

Throws:
```
java.lang.SecurityException - if the caller does not have
```
MonitorAppPermission("handler.recording").
```
java.lang.IllegalArgumentException - if the parameter is null.
```


setPrioritization

```
public abstract void
setPrioritization(org.ocap.resource.ResourceUsage[] resourceUsageList)
```
Sets the relative priorities for a set of ResourceUsages. This method MAY be used by an application with MonitorAppPermission("handler.recording") to set the relative priorities for a set of overlapping resource usages. The implementation SHOULD use the specified prioritization scheme to resolve conflicts (resource conflicts as well as conflicts for RecordingRequests) between these overlapping resource usages. This call MAY change the relative priorities specified by the contention handler or a previous call to this method. Changing the relative priorities for the resource usages MAY result in one or more recording requests changing states. The implementation SHALL only change the ordering of the ResourceUsages passed in the resourceUsageList parameter. This method is meant to be used with the getPrioritizedResourceUsages method.

Parameters:
`resourceUsageList` - a list of ResourceUsages sorted in descending order of prioritization

Throws:
```
java.lang.SecurityException - if the caller does not have
```
MonitorAppPermission("handler.recording").
```
java.lang.IllegalArgumentException - if the parameter does not match a current set of
```
overlapping ResourceUsages.

requestBuffering

```
public abstract void requestBuffering(BufferingRequest request)
```
Requests the implementation to start buffering a service using implementation specific time-shift storage. If successful, the service will be buffered, but audio and video presentation will not take place.

Parameters:
`request` - The to make active.

Throws:

`java.lang.SecurityException` - if the calling application does not have the "file" element set to true in its permission request file.

getBufferingRequests

`public abstract BufferingRequest[] `**`getBufferingRequests`**`()`
> Gets a set of buffering requests that were passed to the requestBuffering method and have not been cancelled.
> **Returns:**
> An array of active buffering requests, or a 0 length array if no buffering requests are active.

cancelBufferingRequest

`public abstract void `**`cancelBufferingRequest`**`(BufferingRequest request)`
> Cancels an active buffering request. If the parameter is not active this method does nothing and returns successfully.
> **Parameters:**
> `request` - The `BufferingRequest` to cancel.
> **Throws:**
> `java.lang.SecurityException` - if the calling application does not have write permission for the request as determined by the `ExtendedFileAccessPermissions` returned by the `getExtendedFileAccessPermissions` method in the parameter, or if the calling application does not have MonitorAppPermission("handler.recording").

getMaxBitRate

`public abstract long `**`getMaxBitRate`**`()`
> Gets the maximum bit rate the implementation will use for duration to space in calculations.
> **Returns:**
> Maximum bit-rate in bits per second.

record

`public abstract org.ocap.shared.dvr.RecordingRequest`
`record``(org.ocap.shared.dvr.RecordingSpec source,`

`java.lang.String[] keys,`

`java.io.Serializable[] appData)`
                                                `throws`
`java.lang.IllegalArgumentException,`

`org.ocap.shared.dvr.AccessDeniedException`
> Records the stream or streams according to the source parameter. The concrete sub-class of RecordingSpec MAY define additional semantics to be applied when instances of that sub-class are used. Overloaded from the `org.ocap.shared.dvr.RecordingManager.record` method. This method is identical to that method except for the key and appData parameters used to add application specific private data.

The keys and appData parameters are parallel arrays where the first entry in the keys array corresponds to the first entry in the appData array and so forth. When a `RecordingRequest` is created from a call to this method and then delivered to a `RecordingChangedListener`, the request SHALL contain the application data passed to this method.

> **Parameters:**
> `source` - specification of stream or streams to be recorded and how they are to be recorded.
> `keys` - the IDs under which the application data is to be added.

appData - the private application data to be added.
Returns:
an instance of RecordingRequest that represents the added recording.
Throws:
java.lang.IllegalArgumentException - if the source is an application defined class or as
defined in the concrete sub-class of RecordingSpec for instances of that class. Also throws this exception if
the keys or appData parameters are null or not the same length.
org.ocap.shared.dvr.AccessDeniedException - if the calling application is not permitted to
perform this operation by RecordingRequest specific security attributes.
java.lang.SecurityException - if the calling application does not have
RecordingPermission("create",..) or RecordingPermission("*",..)

setRecordingDelay

public abstract void **setRecordingDelay**(long seconds)
Sets the amount of time to delay the start of scheduled recordings after the initial monitor application is
running. Calling this method more than once over-writes the previous setting.
Parameters:
seconds - Number of seconds to delay.
Throws:
java.lang.SecurityException - if the calling application does not have
MonitorAppPermission("recording").
java.lang.IllegalArgumentException - is the parameter is negative.

signalRecordingStart

public abstract void **signalRecordingStart**()
Informs the implementation it SHALL start scheduled recordings if it hasn't already done so. Terminates
timeout of the delay set by the #setRecordingDelay method if it is still in effect.
Throws:
java.lang.SecurityException - if the calling application does not have
MonitorAppPermission("recording").

getSmallestTimeShiftDuration

public abstract long **getSmallestTimeShiftDuration**()
Gets the smallest time-shift duration supported by the implementation. This method SHALL return a value
greater than zero.
Returns:
The smallest time-shift duration in seconds that is supported by the implementation.

enableBuffering

public abstract void **enableBuffering**()
Enables time-shift buffering and buffering without presentation. The default is buffering is enabled.
Throws:
java.lang.SecurityException - if the calling application does not have
MonitorAppPermission("recording").

disableBuffering

public abstract void **disableBuffering**()
Disables time-shift buffering and buffering without presentation. All time-shift operations cease
immediately and any presenting services that are time-shifted SHALL be taken to the live point. Any

buffering without presentation activities SHALL cease to be honored. Any content in a time-shift buffer before this method was called SHALL not be accessible if the `enableBuffering` method is called. If an implementation uses time-shift buffering for recording creation it MAY segment the recording.

Throws:

`java.lang.SecurityException` - if the calling application does not have MonitorAppPermission("recording").

org.ocap.dvr
Class OcapRecordingProperties

```
java.lang.Object
  └─org.ocap.shared.dvr.RecordingProperties
      └─org.ocap.dvr.OcapRecordingProperties
```

```
public class OcapRecordingProperties
extends RecordingProperties
```

Encapsulates the details about how a recording is to be made.

For purposes of the `setRecordingProperties` method, properties MAY be changed under the following state conditions:

- bitRate - in the PENDING_NO_CONFLICT_STATE and PENDING_WITH_CONFLICT_STATE

- priorityFlag - in the PENDING_NO_CONFLICT_STATE and PENDING_WITH_CONFLICT_STATE

- retentionPriority - any state except DELETED_STATE and CANCELLED_STATE

- access - any state

- organization - cannot be changed in any state

- destination - in the PENDING_NO_CONFLICT_STATE and PENDING_WITH_CONFLICT_STATE

- expirationPeriod - any state except DELETED_STATE and CANCELLED_STATE

- resourcePriority - any state except DELETED_STATE, CANCELLED_STATE, FAILED_STATE, COMPLETE_STATE, or INCOMPLETE_STATE

Field Summary

static int	**DELETE_AT_EXPIRATION** Indicates a recording SHALL be deleted by the implementation as soon as its expiration date is reached.
static byte	**HIGH_BIT_RATE** Indicates an implementation specific value for high bit-rate.
static byte	**LOW_BIT_RATE** Indicates an implementation specific value for low bit-rate.
static byte	**MEDIUM_BIT_RATE** Indicates an implementation specific value for medium bit-rate.
static byte	**RECORD_IF_NO_CONFLICTS** Record only if there are no conflicts.
static byte	**RECORD_WITH_CONFLICTS** Record even when resource conflicts exist.
static byte	**TEST_RECORDING** Schedule only test recording requests corresponding to this spec.

Constructor Summary

OcapRecordingProperties(byte bitRate, long expirationPeriod, int retentionPriority, byte priorityFlag, ExtendedFileAccessPermissions access, java.lang.String organization, MediaStorageVolume destination)   Constructs an immutable instance of OcapRecordingProperties with the specified attributes.
OcapRecordingProperties(byte bitRate, long expirationPeriod, int retentionPriority, byte priorityFlag, ExtendedFileAccessPermissions access, java.lang.String organization, MediaStorageVolume destination, int resourcePriority)   Constructs an immutable instance of OcapRecordingProperties with the specified attributes.

Method Summary

ExtendedFileAccessPermissions	**getAccessPermissions**() Return the file access permission to use for the recording
byte	**getBitRate**() Return the bitRate to use for the recording
MediaStorageVolume	**getDestination**() Return the volume that represents the storage location of the recording
long	**getExpirationPeriod**() Gets the period in seconds the recording expires after being scheduled.
java.lang.String	**getOrganization**() Return the name of the organization that this recording will be tied to
byte	**getPriorityFlag**() Return whether or not the recording should be made if there are resource conflicts
int	**getResourcePriority**() Return the application-specified resource priority that may be considered at resource contention resolution time.
int	**getRetentionPriority**() Gets the priority determining how the recording is deleted.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

HIGH_BIT_RATE

public static final byte **HIGH_BIT_RATE**
        Indicates an implementation specific value for high bit-rate.

LOW_BIT_RATE

public static final byte **LOW_BIT_RATE**
> Indicates an implementation specific value for low bit-rate.

MEDIUM_BIT_RATE

public static final byte **MEDIUM_BIT_RATE**
> Indicates an implementation specific value for medium bit-rate.

DELETE_AT_EXPIRATION

public static final int **DELETE_AT_EXPIRATION**
> Indicates a recording SHALL be deleted by the implementation as soon as its expiration date is reached.

RECORD_IF_NO_CONFLICTS

public static final byte **RECORD_IF_NO_CONFLICTS**
> Record only if there are no conflicts.

RECORD_WITH_CONFLICTS

public static final byte **RECORD_WITH_CONFLICTS**
> Record even when resource conflicts exist.

TEST_RECORDING

public static final byte **TEST_RECORDING**
> Schedule only test recording requests corresponding to this spec. Does not cause a recording to be started. This value could be used as the priorityFlag parameter value to the constructor for instances of this class. When an OcapRecordingProperties with this value used as a priority value is used to schedule a recording request, any leaf recording requests scheduled will be in the TEST_STATE. If a test recording request is unresolved, partially resolved or completely resolved, the states would be UNRESOLVED_STATE, PARTIALLY_RESOLVED_STATE and COMPLETELY_RESOLVED_STATE respectively. Test recording requests maybe used by applications to detect potential conflicts before scheduling a regular recording. Scheduling a test recording request will not affect the states of any other recording requests. No events will be generated corresponding to a test recording request. Test recording requests will not change state to any other state.

Constructor Detail

OcapRecordingProperties

public **OcapRecordingProperties**(byte bitRate,
                                   long expirationPeriod,
                                   int retentionPriority,
                                   byte priorityFlag,
                                   ExtendedFileAccessPermissions access,
                                   java.lang.String organization,
                                   MediaStorageVolume destination)
                        throws java.lang.IllegalArgumentException
> Constructs an immutable instance of OcapRecordingProperties with the specified attributes.

The unspecified resourcePriority attribute is assigned a value of PRIORITY_UNKNOWN.

> **Parameters:**

bitRate - An application may specify LOW_BIT_RATE, MEDIUM_BIT_RATE, or HIGH_BIT_RATE. For analog recordings the corresponding bit-rate values are implementation specific. For digital recordings these values request optional transrating. When transrating is supported, HIGH_BIT_RATE indicates no transrating, and MEDIUM_BIT_RATE to LOW_BIT_RATE indicates increasing compression with a potential decrease in video quality.

expirationPeriod - The period in seconds after the initiation of recording when leaf recording requests with this recording property are deemed as expired. The implementation will delete recorded services based on the expirationPeriod and retentionPriority parameters. This is done without application intervention and transitions those recording requests to the deleted state.

retentionPriority - Indicates when the recording shall be deleted. An application MAY pass in DELETE_AT_EXPIRATION or a higher value indicating a retention priority. If the value is not DELETE_AT_EXPIRATION the recording will be kept after the expirationPeriod has passed if the implementation does not need the storage space for any other reason. If the space is needed expired recordings will be deleted based on retention priority, i.e. higher value equals higher priority, until the needed space is achieved.

priorityFlag - Indication whether the recording should be made regardless of resource conflict or not. This parameter can contain the values RECORD_IF_NO_CONFLICTS, TEST_RECORDING or RECORD_WITH_CONFLICTS.

access - File access permission for the recording request.

organization - Name of the organization this recording will be tied to. This String will be compared against the organization_id as would be found in the organization name field of an application's leaf certificate to authenticate a playback request. A value of null disables such playback authentication for this recording.

destination - The volume that represents the Storage location of the recording. When an instance of this class is used with a ServiceRecordingSpec a LocatorRecordingSpec, or a ServiceContextRecordingSpec where the specified service context is not attached to a time-shift buffer, with the value of this parameter set to null, the implementation shall use the default recording volume (see org.ocap.storage.MediaStorageOption) in one of the storage devices connected. If the value is null when used with a ServiceContextRecordingSpec, when the service context specified in the ServiceContextRecordingSpec is attached to a time-shift buffer, the default recording volume from the storage device where the time-shift buffer is located shall be used. When an instance of this class is used with a ServiceContextRecordingSpec, the record(..) method will throw an IllegalArgumentException if the destination is not in same storage device where an attached time-shift buffer is located.

Throws:

java.lang.IllegalArgumentException - if bitRate does not equal one of LOW_BIT_RATE, MEDIUM_BIT_RATE, or HIGH_BIT_RATE; or if priorityFlag does not contain the value RECORD_IF_NO_CONFLICTS, TEST_RECORDING or RECORD_WITH_CONFLICTS; or if organization is not found in the application's certificate file.

OcapRecordingProperties

```
public OcapRecordingProperties(byte bitRate,
                               long expirationPeriod,
                               int retentionPriority,
                               byte priorityFlag,
                               ExtendedFileAccessPermissions access,
                               java.lang.String organization,
                               MediaStorageVolume destination,
                               int resourcePriority)
                      throws java.lang.IllegalArgumentException
```

Constructs an immutable instance of OcapRecordingProperties with the specified attributes.

Parameters:

bitRate - An application may specify LOW_BIT_RATE, MEDIUM_BIT_RATE, or HIGH_BIT_RATE. For analog recordings the corresponding bit-rate values are implementation specific. For digital recordings these values request optional transrating. When transrating is supported,

HIGH_BIT_RATE indicates no transrating, and MEDIUM_BIT_RATE to LOW_BIT_RATE indicates increasing compression with a potential decrease in video quality.

`expirationPeriod` - The period in seconds after the initiation of recording when leaf recording requests with this recording property are deemed as expired. The implementation will delete recorded services based on the expirationPeriod and retentionPriority parameters. This is done without application intervention and transitions those recording requests to the deleted state.

`retentionPriority` - Indicates when the recording shall be deleted. An application MAY pass in DELETE_AT_EXPIRATION or a higher value indicating a retention priority. If the value is not DELETE_AT_EXPIRATION the recording will be kept after the expirationPeriod has passed if the implementation does not need the storage space for any other reason. If the space is needed expired recordings will be deleted based on retention priority, i.e. higher value equals higher priority, until the needed space is achieved.

`priorityFlag` - Indication whether the recording should be made regardless of resource conflict or not. This parameter can contain the values RECORD_IF_NO_CONFLICTS, TEST_RECORDING or RECORD_WITH_CONFLICTS.

`access` - File access permission for the recording request.

`organization` - Name of the organization this recording will be tied to. This `String` will be compared against the `organization_id` as would be found in the organization name field of an application's leaf certificate to authenticate a playback request. A value of null disables such playback authentication for this recording.

`destination` - The volume that represents the Storage location of the recording. When an instance of this class is used with a ServiceRecordingSpec a LocatorRecordingSpec, or a ServiceContextRecordingSpec where the specified service context is not attached to a time-shift buffer, with the value of this parameter set to null, the implementation shall use the default recording volume (see org.ocap.storage.MediaStorageOption) in one of the storage devices connected. If the value is null when used with a ServiceContextRecordingSpec, when the service context specified in the ServiceContextRecordingSpec is attached to a time-shift buffer, the default recording volume from the storage device where the time-shift buffer is located shall be used. When an instance of this class is used with a ServiceContextRecordingSpec, the record(..) method will throw an IllegalArgumentException if the destination is not in same storage device where an attached time-shift buffer is located.

`resourcePriority` - Indicates the application-specified resource priority that should be used for any implied resource reservations. This value will be made available for consideration for resource contention resolution.

Throws:

`java.lang.IllegalArgumentException` - if bitRate does not equal one of LOW_BIT_RATE, MEDIUM_BIT_RATE, or HIGH_BIT_RATE; or if priorityFlag does not contain the value RECORD_IF_NO_CONFLICTS, TEST_RECORDING or RECORD_WITH_CONFLICTS; or if organization is not found in the application's certificate file; or if resourcePriority is not a valid resource priority as defined by the `ResourcePriority` interface.

Method Detail

getBitRate

`public byte` **`getBitRate`**`()`

Return the bitRate to use for the recording

Returns:

the bitRate as passed into the constructor

getExpirationPeriod

`public long` **`getExpirationPeriod`**`()`

Gets the period in seconds the recording expires after being scheduled.

Overrides:

getExpirationPeriod in class RecordingProperties
Returns:
the expiration period as passed into the constructor

getRetentionPriority

public int **getRetentionPriority**()
Gets the priority determining how the recording is deleted.
Returns:
the retention priority as passed into the constructor

getPriorityFlag

public byte **getPriorityFlag**()
Return whether or not the recording should be made if there are resource conflicts
Returns:
the priority flag passed into the constructor

getAccessPermissions

public ExtendedFileAccessPermissions **getAccessPermissions**()
Return the file access permission to use for the recording
Returns:
the file access permission passed into the constructor

getOrganization

public java.lang.String **getOrganization**()
Return the name of the organization that this recording will be tied to
Returns:
the organization passed into the constructor

getDestination

public MediaStorageVolume **getDestination**()
Return the volume that represents the storage location of the recording
Returns:
the volume passed into the constructor

getResourcePriority

public int **getResourcePriority**()
Return the application-specified resource priority that may be considered at resource contention resolution time.
Returns:
the resource priority

org.ocap.dvr
Interface OcapRecordingRequest

All Superinterfaces:
        LeafRecordingRequest, RecordingRequest

```
public interface OcapRecordingRequest
extends LeafRecordingRequest
```

This interface represents a LeafRecordingRequest in OCAP.

When the implementation detects a schedule conflict, it either resolves the conflict using the Application priority of the conflicting recordings, or invokes the `ResourceContentionHandler` if one is set. The resolution of the conflict by the implementation or the ResourceContentionHandler will result in some of the overlapping recordings to be pending without conflict and some to be pending with conflict.

Field Summary

static int	**CANCELLED_STATE** This recording request was cancelled.
static int	**TEST_STATE** This recording request is a test recording request.

Fields inherited from interface org.ocap.shared.dvr.LeafRecordingRequest
COMPLETED_STATE, DELETED_STATE, FAILED_STATE, IN_PROGRESS_INCOMPLETE_STATE, IN_PROGRESS_INSUFFICIENT_SPACE_STATE, IN_PROGRESS_STATE, IN_PROGRESS_WITH_ERROR_STATE, INCOMPLETE_STATE, PENDING_NO_CONFLICT_STATE, PENDING_WITH_CONFLICT_STATE

Method Summary

void	**cancel**() Cancels a pending recording request.
RecordingList	**getOverlappingEntries**() Gets any other RecordingRequest that overlaps with the duration of this recording request.
long	**getSpaceRequired**() Gets the estimated space, in bytes, required for the recording.
boolean	**isStorageReady**() Returns whether the destined `MediaStorageVolume` for this recording is present and ready or not.

Methods inherited from interface org.ocap.shared.dvr.LeafRecordingRequest
getDeletionDetails, getFailedException, getService, stop

Methods inherited from interface org.ocap.shared.dvr.RecordingRequest

addAppData, delete, getAppData, getAppID, getId, getKeys, getParent,
getRecordingSpec, getRoot, getState, isRoot, removeAppData, reschedule,
setRecordingProperties

Field Detail

TEST_STATE

static final int **TEST_STATE**
> This recording request is a test recording request. Actual recording is not initiated for recording requests in this state. RecordingRequests in this state do not transition to other states. No events are generated when a recording request is added or deleted in this state. A recording in this state is a leaf recording request. Recordings in this states are the leaf recording requests corresponding to invocation of the RecordingManager.record(..) method with the priority value in OcapRecordingProperties set to TEST_RECORDING.

CANCELLED_STATE

static final int **CANCELLED_STATE**
> This recording request was cancelled. Transitioned to when the cancel method completes successfully.

Method Detail

cancel

void **cancel**()
>   throws java.lang.IllegalStateException,
>   AccessDeniedException
>
> Cancels a pending recording request. The recording request will be not be deleted from the database after the successful invocation of this method. Successful completion places this recording request in the CANCELLED_STATE. Canceling a recording request may resolve one or more conflicts. In this case some pending recordings with conflicts would be changed to pending without conflicts.
> **Specified by:**
> cancel in interface LeafRecordingRequest
> **Throws:**
> AccessDeniedException - if the calling application is not permitted to perform this operation by RecordingRequest specific security attributes.
> java.lang.SecurityException - if the calling application does not have RecordingPermission("cancel",..) or RecordingPermission("*",..)
> java.lang.IllegalStateException - if the state of the recording is not in PENDING_STATE_NO_CONFLICT_STATE or PENDING_WITH_CONFLICT_STATE.

getSpaceRequired

long **getSpaceRequired**()
> Gets the estimated space, in bytes, required for the recording.
> **Returns:**
> Space required for the recording in bytes. This method returns zero if the recordings is in failed state.

---

getOverlappingEntries

RecordingList **getOverlappingEntries**()

> Gets any other RecordingRequest that overlaps with the duration of this recording request. This method will return null unless the recording request is in the PENDING_WITH_CONFLICTS_STATE, PENDING_NO_CONFLICTS_STATE, IN_PROGRESS_INSUFFICIENT_SPACE_STATE or IN_PROGRESS_STATE. The returned list will contain only overlapping recording requests for which the application has read access permission. The RecordingList returned is only a copy of the list of overlapping entries at the time of this method call. This list is not updated if there are any changes. A new call to this method will be required to get the updated list.
>
> **Returns:**
> a RecordingList

isStorageReady

boolean **isStorageReady**()

> Returns whether the destined `MediaStorageVolume` for this recording is present and ready or not. This method SHALL return `true` under the following conditions:

- `getRecordingProperties().getDestination().getStatus() == READY`

- `getRecordingProperties().getDestination()` returns null and a default recording volume exists for which `getStatus() == READY`

> Otherwise, `false` SHALL be returned.
>
> **Returns:**
> `true` if the explicit or an implicit destination volume is present and ready; `false` otherwise

org.ocap.dvr
Class PrivateRecordingSpec

```
java.lang.Object
  └─org.ocap.shared.dvr.RecordingSpec
       └─org.ocap.dvr.PrivateRecordingSpec
```

public class **PrivateRecordingSpec**
extends org.ocap.shared.dvr.RecordingSpec

Specifies a recording request that can be resolved only by an application defined request resolution handler.

Constructor Summary

PrivateRecordingSpec(java.io.Serializable requestData,
org.ocap.shared.dvr.RecordingProperties properties)
    Constructor

Method Summary

java.io.Serializable	**getPrivateData**() Returns the private data stored in this recording spec

Methods inherited from class org.ocap.shared.dvr.RecordingSpec

getProperties

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,
wait, wait, wait

Constructor Detail

PrivateRecordingSpec

public **PrivateRecordingSpec**(java.io.Serializable requestData,

org.ocap.shared.dvr.RecordingProperties properties)
      Constructor
      Parameters:
      requestData - private data the format of which is known only to the application.
      properties - the definition of how the recording is to be done

Method Detail

getPrivateData

public java.io.Serializable **getPrivateData**()
      Returns the private data stored in this recording spec
      Returns:

the private data passed into the constructor

org.ocap.dvr
Class RecordingAlertEvent

```
java.lang.Object
  └java.util.EventObject
      └org.ocap.dvr.RecordingAlertEvent
```
All Implemented Interfaces:
       java.io.Serializable

```
public class RecordingAlertEvent
extends java.util.EventObject
```

Event notifying that a scheduled recording is about to occur. This event is triggered for
LeafRecordingRequests in pending states.

Field Summary

Fields inherited from class java.util.EventObject
source

Constructor Summary

RecordingAlertEvent(org.ocap.shared.dvr.RecordingRequest source)
Constructs the event.

Method Summary

org.ocap.shared.dvr.RecordingRequest	**getRecordingRequest**() Returns the RecordingRequest that caused the event.

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

RecordingAlertEvent

```
public RecordingAlertEvent(org.ocap.shared.dvr.RecordingRequest source)
```
       Constructs the event.
       Parameters:
       source - The RecordingRequest that caused the event.

Method Detail

getRecordingRequest

```
public org.ocap.shared.dvr.RecordingRequest getRecordingRequest()
```
> Returns the RecordingRequest that caused the event.
> **Returns:**
> The RecordingRequest that caused the event.

org.ocap.dvr
Interface RecordingAlertListener

All Superinterfaces:
> java.util.EventListener

public interface **RecordingAlertListener**
extends java.util.EventListener

Listener for Recording Alerts.

Method Summary

void	**recordingAlert**(RecordingAlertEvent e) Notifies the RecordingAlertListener that a scheduled activity is about to happen.

Method Detail

recordingAlert

void **recordingAlert**(RecordingAlertEvent e)
> Notifies the RecordingAlertListener that a scheduled activity is about to happen.
> **Parameters:**
> e - The generated event.

org.ocap.dvr
Interface RecordingPlaybackListener

All Superinterfaces:
>   java.util.EventListener

public interface **RecordingPlaybackListener**
extends java.util.EventListener

This interface represents a listener that can be added to listen for recording playback start. The implementation SHALL notify a listener once when a recording playback starts. For purposes of this listener playback is considered ongoing while the presenting ServiceContext is in the presenting state regardless of trick mode. This listener is specific to ServiceContext recording playback and does not notify for discreet Player based recording playback.

Method Summary

void	**notifyRecordingPlayback** (javax.tv.service.selection.ServiceContext context, int artificialCarouselID, int[] carouselIDs) Notifies the listener a recording playback has started.

Method Detail

notifyRecordingPlayback

void
notifyRecordingPlayback(javax.tv.service.selection.ServiceContext context,
                                int artificialCarouselID,
                                int[] carouselIDs)

>   Notifies the listener a recording playback has started. The implementation SHALL create a new carousel Id for any artificial carousel in each playback. The carouselIDs parameter SHALL reference broadcast carousels when stored with a recorded service. An artificial carousel ID shall not conflict with a carousel ID of a signaled carousel that was also stored with the recorded service and presented by the context parameter. An artificial carousel ID MAY conflict with other carousel IDs.
>   **Parameters:**
>   context - The ServiceContext presenting the recorded service.
>   artificialCarouselID - Carousel ID for an artificial carousel that MAY have been created for the recording being played back. A value of -1 indicates no artificial carousel was created.
>   carouselIDs - Array of carousel IDs associated with broadcast carousels stored with the recording being played back. If no carousels are contained a zero length array is passed in.

org.ocap.dvr
Interface RecordingResourceUsage

All Superinterfaces:
>   org.ocap.resource.ResourceUsage

```
public interface RecordingResourceUsage
extends org.ocap.resource.ResourceUsage
```

This interface represents a grouping of resources specific to an recording function performed by an application.

Method Summary

org.ocap.shared.dvr.RecordingRequest	**getRecordingRequest**() Gets the RecordingRequest associated with the set of resources contained in the usage and initiated by the application returned by the base ResourceUsage.getAppID method.

Methods inherited from interface org.ocap.resource.ResourceUsage
getAppID, getResource, getResourceNames

Method Detail

getRecordingRequest

org.ocap.shared.dvr.RecordingRequest **getRecordingRequest**()
>   Gets the RecordingRequest associated with the set of resources contained in the usage and initiated
>   by the application returned by the base ResourceUsage.getAppID method.
>   **Returns:**
>   The recording request associated with the resource usage.

org.ocap.dvr
Interface RequestResolutionHandler

public interface **RequestResolutionHandler**

This interface will be implemented by the application that registers the RequestResolutionHandler. The RequestResolutionHandler will be invoked whenever a new unresolved recording request is added to the RecordingManager database. The RecordingResolutionHandler may call the resolve(..) method of the OcapRecordingManager multiple times to schedule one or more recording requests corresponding to the recording request.

Method Summary

void	**requestResolution**(org.ocap.shared.dvr.RecordingRequest request) This method would be invoked by the implementation when an unresolved recording request is scheduled in response to an application calling the record(..) method of the RecordingManager.

Method Detail

requestResolution

void **requestResolution**(org.ocap.shared.dvr.RecordingRequest request)
> This method would be invoked by the implementation when an unresolved recording request is scheduled in response to an application calling the record(..) method of the RecordingManager.

org.ocap.dvr
Interface SharedResourceUsage

All Superinterfaces:
>   ResourceUsage, SharedResourceUsage

```
public interface SharedResourceUsage
extends SharedResourceUsage
```

This interface represents a group of resources where one or more resources are shared between multiple resource usages. For example, when a tuner is used for an ongoing recording and also for presenting a broadcast service in a service context, and if the tuner is in a resource contention, the tuner is considered shared between a `RecordingResourceUsage` and a `ServiceContextResourceUsage`. If there is a resource contention for a tuner, the shared usage of tuner is represented by a `SharedResourceUsage` where the getResourceUsages() method would return both `ResourceUsage` instances that share the tuner.

Because a `SharedResourceUsage` can contain multiple `ResourceUsage` instances where different applications reserved the resources, the value returned by the `SharedResourceUsage.getAppID` method is meaningless and SHALL be null. To determine AppID instances an application can peruse the `ResourceUsage` instances returned by the `getResourceUsages` method.

Method Summary

ResourceUsage[]	**getResourceUsages**() Gets the list of `ResourceUsage` instances that share the resources represented by this resource usage.
ResourceUsage[]	**getResourceUsages**(ResourceProxy resource) Gets the list of `ResourceUsage` instances that share a particular resource.

Methods inherited from interface org.ocap.resource.ResourceUsage
getAppID, getResource, getResourceNames, getResourcePriority

Method Detail

getResourceUsages

ResourceUsage[] **getResourceUsages**()
>   Gets the list of `ResourceUsage` instances that share the resources represented by this resource usage.
>   **Specified by:**
>   getResourceUsages in interface SharedResourceUsage
>   **Returns:**
>   An array of `ResourceUsage` instances that share one or more resources.

getResourceUsages

ResourceUsage[] **getResourceUsages**(ResourceProxy resource)
>   Gets the list of `ResourceUsage` instances that share a particular resource.
>   **Specified by:**
>   getResourceUsages in interface SharedResourceUsage
>   **Parameters:**
>   resource - The shared resource for which `ResourceUsage` instances should be returned.
>   **Returns:**

An array of `ResourceUsage` instances that share the specified resource.

org.ocap.dvr
Interface TimeShiftBufferResourceUsage

All Superinterfaces:
> org.ocap.resource.ResourceUsage

public interface **TimeShiftBufferResourceUsage**
extends org.ocap.resource.ResourceUsage

This interface represents a grouping of resources specific to a time-shift buffering performed by an application.

Method Summary

javax.tv.service.Service	**getService**() Gets the `Service` associated with the set of resources contained in the usage where the last service selection was initiated by the application returned by the base ResourceUsage.getAppID method.

Methods inherited from interface org.ocap.resource.ResourceUsage
getAppID, getResource, getResourceNames

Method Detail

getService

javax.tv.service.Service **getService**()
> Gets the `Service` associated with the set of resources contained in the usage where the last service selection was initiated by the application returned by the base ResourceUsage.getAppID method.
> **Returns:**
> The Service associated with the resource usage.

org.ocap.dvr
Class TimeShiftEvent

```
java.lang.Object
  └ java.util.EventObject
      └ javax.tv.service.selection.ServiceContextEvent
          └ org.ocap.dvr.TimeShiftEvent
```

All Implemented Interfaces:
    java.io.Serializable

```
public class TimeShiftEvent
extends javax.tv.service.selection.ServiceContextEvent
```

The parent class for TimeShiftBuffer events.

Field Summary

static int	**NO_TIME_SHIFT_BUFFER** A time-shift buffer or recording was not found for attachment to the ServiceContext
static int	**TIME_SHIFT_BUFFER_FOUND** A time-shift buffer or recording was found for attachment to the ServiceContext.
static int	**TIME_SHIFT_PROPERTIES_CHANGED** The implementation was forced to change time-shift properties due to signaling.

Fields inherited from class java.util.EventObject
source

Constructor Summary

TimeShiftEvent(javax.tv.service.selection.ServiceContext source, int reason)   Constructor for this event.

Method Summary

int	**getReason**() Gets the reason for this event.
javax.tv.service.selection.ServiceContext	**getServiceContext**() Reports the ServiceContext that generated the event.

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

TIME_SHIFT_BUFFER_FOUND

public static final int **TIME_SHIFT_BUFFER_FOUND**
> A time-shift buffer or recording was found for attachment to the ServiceContext.

NO_TIME_SHIFT_BUFFER

public static final int **NO_TIME_SHIFT_BUFFER**
> A time-shift buffer or recording was not found for attachment to the ServiceContext

TIME_SHIFT_PROPERTIES_CHANGED

public static final int **TIME_SHIFT_PROPERTIES_CHANGED**
> The implementation was forced to change time-shift properties due to signaling.

Constructor Detail

TimeShiftEvent

public **TimeShiftEvent**(javax.tv.service.selection.ServiceContext source,
                        int reason)
> Constructor for this event.
> **Parameters:**
> source - The object associated with this event.
> reason - The reason code for this event. See constants in this class for possible values.
> **Throws:**
> java.lang.IllegalArgumentException - if the reason code is not a value matching one of the
> possible constants.

Method Detail

getReason

public int **getReason**()
> Gets the reason for this event.
> **Returns:**
> The reason code for this event. See constants in this class for possible return values; see constants in this
> class.

getServiceContext

public javax.tv.service.selection.ServiceContext **getServiceContext**()
> Reports the ServiceContext that generated the event.
> **Overrides:**
> getServiceContext in class javax.tv.service.selection.ServiceContextEvent
> **Returns:**
> The ServiceContext that generated the event.

---

org.ocap.dvr
Interface TimeShiftListener

All Superinterfaces:
> java.util.EventListener

```
public interface TimeShiftListener
extends java.util.EventListener
```

The TimeShiftListener interface is implemented by applications wishing to receive events related to time shift resources.

Method Summary

void	**receiveTimeShiftevent**(TimeShiftEvent e) Notifies the TimeShiftListener of an event generated by time-shift resource handling.

Method Detail

receiveTimeShiftevent

```
void receiveTimeShiftevent(TimeShiftEvent e)
```
>   Notifies the TimeShiftListener of an event generated by time-shift resource handling.
>   **Parameters:**
>   e - The generated event.

org.ocap.dvr
Interface TimeShiftProperties

public interface **TimeShiftProperties**

This interface represents a set of time-shift properties that can be set for and queried from a `ServiceContext`. Any Host device that supports the OpenCable DVR extension SHALL implement this interface by any class that also implements the ServiceContext interface.

Method Summary

void	**addTimeShiftListener**(TimeShiftListener listener) Adds a listener for time-shift events related to this `TimeShiftProperties`.
boolean	**getLastServiceBufferedPreference**() Gets the "last" service buffered preference.
long	**getMaximumDuration**() Gets the maximum content buffering duration.
long	**getMinimumDuration**() Gets the minimum content buffering duration.
boolean	**getSavePreference**() Gets the save time-shift contents at service change preference.
void	**removeTimeShiftListener**(TimeShiftListener listener) Removes a previously added listener for time-shift events from this `TimeShiftProperties`.
void	**setLastServiceBufferedPreference**(boolean buffer) Sets a preference to buffer the last service.
void	**setMaximumDuration**(long maxDuration) Sets the maximum duration of content that MAY be buffered for this `ServiceContext`.
void	**setMinimumDuration**(long minDuration) Sets the minimum duration of content that SHALL be buffered for this `ServiceContext`.
void	**setPresentation**(javax.tv.service.Service service, javax.media.Time time, float rate, boolean action, boolean persistent) Sets the JMF media time location from where the playback will begin when a specific service is selected with this service context.
void	**setSavePreference**(boolean save) Sets a preference to retain the time-shift contents for the `ServiceContext` when a new service is selected.

Method Detail

addTimeShiftListener

void **addTimeShiftListener**(TimeShiftListener listener)
          Adds a listener for time-shift events related to this `TimeShiftProperties`.

Parameters:
`listener` - The listener to add.

removeTimeShiftListener

void **removeTimeShiftListener**(TimeShiftListener listener)

Removes a previously added listener for time-shift events from this `TimeShiftProperties`. If the given listener has not previously been added then this method has no effect.
Parameters:
`listener` - The listener to remove.

getMinimumDuration

long **getMinimumDuration**()

Gets the minimum content buffering duration. If this method is called before setMinimumDuration has ever been called, or if content buffering is disabled for this `ServiceContext` the value returned SHALL be 0.
Returns:
The minimum content buffering duration in seconds.

setMinimumDuration

void **setMinimumDuration**(long minDuration)

Sets the minimum duration of content that SHALL be buffered for this `ServiceContext`. Setting the minimum duration to 0 disables time shifting on the ServiceContext.

This method MAY be called at any time regardless of service context state. However, enabling time-shifting or changing the minimum duration SHALL NOT take affect until the `ServiceContext` is in the not presenting state, presentation pending state, or a new service is selected. If the same service is selected it is implementation dependent regarding whether time-shift enabling takes affect during the selection.

Disabling time shifting by setting the minimum duration to 0 SHOULD take effect immediately.

When enabling of time shifting by changing the minimum duration from zero to a positive value takes effect, a TimeShiftControl SHALL be added to the associated JMF player. When time shifting is disabled by changing the minimum duration to zero any existing TimeShiftControl SHALL be removed from the associated JMF player.

An increase in minimum duration MUST NOT cause any loss of previously buffered content for the current service.

Parameters:
`minDuration` - Minimum duration in seconds.
Throws:
`java.lang.IllegalArgumentException` - If the parameter is greater than the current value and Host device does not have enough space to meet the request, or if the parameter is greater than the maximum duration set by the `setMaximumDuration` method, or if the parameter is less than the duration returned by `OcapRecordingManager.getSmallestTimeShiftDuration()`.
`java.lang.SecurityException` - if the calling application does not have `ServiceContextPermission("*","own")` for the `ServiceContext` object that implements this `TimeShiftProperties`.

getMaximumDuration

long **getMaximumDuration**()

Gets the maximum content buffering duration. If this method is called before `setMaximumDuration` has ever been called, or if content buffering is disabled for this `ServiceContext` the value returned SHALL be 0.

Returns:
The maximum content buffering duration in seconds.

setMaximumDuration

void **setMaximumDuration**(long maxDuration)
Sets the maximum duration of content that MAY be buffered for this ServiceContext. Informs the implementation that storing more content than this is not needed by the application owning this ServiceContext.

This method MAY be called at any time regardless of service context state.

Parameters:
maxDuration - Maximum duration in seconds.
Throws:
java.lang.IllegalArgumentException - if the parameter is less than the duration set by the setMinimumDuration method, or if the parameter is less than the duration returned by OcapRecordingManager.getSmallestTimeShiftDuration().
java.lang.SecurityException - if the calling application does not have ServiceContextPermission("*","own") for the ServiceContext object that implements this TimeShiftProperties.

getLastServiceBufferedPreference

boolean **getLastServiceBufferedPreference**()
Gets the "last" service buffered preference.
Returns:
Preference indication for recording the "last" service. Returns true if "last" service should be buffered, otherwise returns false.

setLastServiceBufferedPreference

void **setLastServiceBufferedPreference**(boolean buffer)
Sets a preference to buffer the last service. This method has no effect if the size of the time-shift buffer associated with the ServiceContext object implementing this interface is set to zero.
Parameters:
buffer - If true the implementation will buffer the service selected by the ServiceContext object implementing this interface, based on time-shift buffer availability; see the OCAP DVR API specification time-shift buffer requirements. If false the last service will not be buffered.
Throws:
java.lang.SecurityException - if the calling application does not have ServiceContextPermission("*","own") for the ServiceContext object that implements this TimeShiftProperties.

getSavePreference

boolean **getSavePreference**()
Gets the save time-shift contents at service change preference.
Returns:
True if save time-shift contents at service selection preference is enabled, otherwise returns false.

setSavePreference

void **setSavePreference**(boolean save)

Sets a preference to retain the time-shift contents for the `ServiceContext` when a new service is selected. When enabled the time-shift contents are saved back to the value returned by the `getMaxTimeShiftDuration` method.

Parameters:

`save` - If true the implementation will retain the time-shift contents for the `ServiceContext` when a new service is selected. If false the time-shift contents are flushed when a new service is selected.

Throws:

`java.lang.IllegalArgumentException` - if the parameter is true and the Host device does not have the hardware resources to support the preference.

`java.lang.SecurityException` - if the calling application does not have `ServiceContextPermission("*","own")` for the `ServiceContext` object that implements this `TimeShiftProperties`.

setPresentation

```
void setPresentation(javax.tv.service.Service service,
                     javax.media.Time time,
                     float rate,
                     boolean action,
                     boolean persistent)
```

Sets the JMF media time location from where the playback will begin when a specific service is selected with this service context. Also sets the rate of that playback. If an instance of Time corresponding to value of 0 nanoseconds, or a negative value is set, the playback will begin at the live point. The default values for the time and rate values is live point and normal playback respectively. Calling this method for the same service multiple times sets the values to the most recent call.

The implementation SHALL NOT allow content to be started in the past and beyond the duration set in this `ServiceContext`, even if content with the time parameter is buffered. In that case presentation SHALL begin at the duration in the past or at the live point as determined by the action parameter.

Parameters:

`service` - The service to set the media time for.

`time` - The time the service presentation will start at.

`rate` - The rate at which to start play back.

`action` - Indicates what to do when the media time is not buffered when the service is selected. If true presentation starts at the beginning of the buffer, otherwise presentation starts at the live point.

`persistent` - If true the time and rate apply to every selection of the service, otherwise they will only apply to the selection following a call to this method. In the latter case, once the values are applied to one service selection they are returned to their default values.

Throws:

`java.lang.SecurityException` - if the calling application does not have `ServiceContextPermission("*","own")` for the `ServiceContext` object that implements this `TimeShiftProperties`.

Annex E   OCAP DVR Storage API (org.ocap.dvr.storage)

Package org.ocap.dvr.storage

Extensions to the OCAP Storage API to support Media Storage and Time-shift buffer.

Interface Summary	
FreeSpaceListener	This interface represents a listener that will be notified when a media volume has reached a specified level of remaining free space.
MediaStorageEvent	This interface SHALL be implemented by `StorageManagerEvents` generated by the implementation that involve a `StorageProxy` which contains `MediaStorageVolumes`.
MediaStorageOption	This interface represents an option object provided by a `StorageProxy` that supports media volumes (`MediaStorageVolume`) that are used by the DVR recording and playback APIs for storing media content.
MediaStorageVolume	This interface represents a media volume on a storage device and is contained within a `StorageProxy`.
SpaceAllocationHandler	A class implementing this interface decides whether requests to allocate storage space should be allowed or not.

org.ocap.dvr.storage
Interface FreeSpaceListener

All Superinterfaces:
> java.util.EventListener

```
public interface FreeSpaceListener
extends java.util.EventListener
```

This interface represents a listener that will be notified when a media volume has reached a specified level of remaining free space.

Method Summary

void	**notifyFreeSpace**() Notifies the listener the remaining free space has reached the level specified by `MediaStorageVolume.addFreeSpaceListener` `(org.ocap.dvr.storage.FreeSpaceListener, int)`.

Method Detail

notifyFreeSpace

```
void notifyFreeSpace()
```
> Notifies the listener the remaining free space has reached the level specified by `MediaStorageVolume.addFreeSpaceListener(org.ocap.dvr.storage.FreeSpaceListener, int)`.

org.ocap.dvr.storage
Interface MediaStorageEvent

public interface **MediaStorageEvent**

This interface SHALL be implemented by StorageManagerEvents generated by the implementation that involve a StorageProxy which contains MediaStorageVolumes.

Method Summary

RecordingList	**getEntries**() Returns the list of scheduled, pending, in-progress, and completed recordings for which a contained MediaStorageVolume is an explicit or implicit destination.

Method Detail

getEntries

RecordingList **getEntries**()
            Returns the list of scheduled, pending, in-progress, and completed recordings for which a contained MediaStorageVolume is an explicit or implicit destination.

Note that this may include recordings which no longer exist (in the case of a delete) or recordings that previously did not exist (in the case of an add).

> **Returns:**
> the list of recordings affected by this event.

org.ocap.dvr.storage
Interface MediaStorageOption

All Superinterfaces:
> org.ocap.storage.StorageOption

```
public interface MediaStorageOption
extends org.ocap.storage.StorageOption
```

This interface represents an option object provided by a `StorageProxy` that supports media volumes
(`MediaStorageVolume`) that are used by the DVR recording and playback APIs for storing media content.

The interface distinguishes between content accessible through the DVR APIs and as general purpose files.
Implementations may store these different type of content in one or more filesystems. This is transparent to an
application. Only the general purpose files are visible through the normal file and directory classes in java.io.

The interface can be used to query the amount of storage the storage proxy has for storing all types of application-
visible content. (Some of the capacity may be reserved for internal system use.)

The interface also supports the initialization of the storage proxy with a specified allocation between the two types.
However, on some implementations, changing the allocations may require filesystems to be destroyed and recreated
which may result in the deletion of all application-visible content associated with the storage proxy, including any
storage volumes. On other implementations, a change in allocations may require some or all content of the type
being reduced to be destroyed. Initialization should be done with extreme caution.

Method Summary

MediaStorageVolume	**allocateMediaVolume**(java.lang.String name, org.ocap.storage.ExtendedFileAccessPermissions fap) Allocates a `MediaStorageVolume`.
long	**getAllocatableMediaStorage**() Gets total allocatable media storage available for all MediaStorageVolume instances.
MediaStorageVolume	**getDefaultRecordingVolume**() Gets the default volume that the implementation setup as the default recording volume for the containing `StorageProxy`.
long	**getPlaybackBandwidth**() Gets the playback bandwidth in bits-per-second when only one playback stream and no record streams are open on the entire storage device.
long	**getRecordBandwidth**() Gets the record bandwidth in bits-per-second when only one record stream and no playback streams are open on the entire storage device.
long	**getTotalGeneralStorageCapacity**() Gets the total capacity of the GPFS available for application use in the storage device.
long	**getTotalMediaStorageCapacity**() Gets the total capacity of the MEDIAFS available for application use in the storage device.

Method Summary

void	**initialize**(long mediafsSize) Initializes the storage device so that the there are at least mediafsSize bytes available for MEDIAFS use.
boolean	**simultaneousPlayAndRecord**() Indicates if the storage device supports simultaneous play and record.

Method Detail

allocateMediaVolume

MediaStorageVolume **allocateMediaVolume**(java.lang.String name,

org.ocap.storage.ExtendedFileAccessPermissions fap)
                                                throws
java.lang.IllegalArgumentException

> Allocates a `MediaStorageVolume`. A media volume can contain multi-media content that may impose I/O bandwidth criteria upon the storage device. The new volume will be owned by the application that allocated it.
>
> **Parameters:**
> name - Name of the new MediaStorageVolume.
> fap - Access permissions of the new MediaStorageVolume.
> **Returns:**
> Allocated volume storage.
> **Throws:**
> java.lang.IllegalArgumentException - if the name does not meet Java 1.1.8 directory naming conventions, or if the type is not supported by the storage device.
> java.lang.SecurityException - if the calling application is unsigned.

getDefaultRecordingVolume

MediaStorageVolume **getDefaultRecordingVolume**()

> Gets the default volume that the implementation setup as the default recording volume for the containing StorageProxy.
> **Returns:**
> Default recording volume for the storage device.

getPlaybackBandwidth

long **getPlaybackBandwidth**()

> Gets the playback bandwidth in bits-per-second when only one playback stream and no record streams are open on the entire storage device.
> **Returns:**
> Playback bandwidth in bits-per-second.

getRecordBandwidth

long **getRecordBandwidth**()

> Gets the record bandwidth in bits-per-second when only one record stream and no playback streams are open on the entire storage device.
> **Returns:**
> Record bandwidth in bits-per-second.

getTotalMediaStorageCapacity

long **getTotalMediaStorageCapacity**()

>   Gets the total capacity of the MEDIAFS available for application use in the storage device.
>   **Returns:**
>   Total audio/video capacity of the storage device.

getAllocatableMediaStorage

long **getAllocatableMediaStorage**()

>   Gets total allocatable media storage available for all MediaStorageVolume instances.
>   **Returns:**
>   Size of allocatable media storage in bytes.

getTotalGeneralStorageCapacity

long **getTotalGeneralStorageCapacity**()

>   Gets the total capacity of the GPFS available for application use in the storage device.
>   **Returns:**
>   Total general purpose capacity of the storage device.

initialize

void **initialize**(long mediafsSize)
                    throws java.lang.IllegalArgumentException,
                           java.lang.IllegalStateException

>   Initializes the storage device so that the there are at least mediafsSize bytes available for MEDIAFS use.
>   The effects of initialization may include the deletion of all application visible content associated with the
>   storage proxy. Calling this method may remove application access to storage on the device for the duration
>   of the call. It may cause the abnormal termination of applications with open files associated with the
>   storage proxy. This method will block until the storage proxy is again ready for use.
>   **Parameters:**
>   mediafsSize - New size of the total MEDIAFS capacity in bytes.
>   **Throws:**
>   java.lang.IllegalArgumentException - if the mediafsSize passed is greater than the sum of
>   what is returned by getTotalGeneralStorageCapacity() and getTotalMediaStorageCapacity().
>   java.lang.IllegalStateException - if the sizes cannot be changed by the implementation for
>   any reason.

simultaneousPlayAndRecord

boolean **simultaneousPlayAndRecord**()

>   Indicates if the storage device supports simultaneous play and record.
>   **Returns:**
>   True if simultaneous play and record is supported, otherwise returns false.

org.ocap.dvr.storage
Interface MediaStorageVolume

All Superinterfaces:
　　　LogicalStorageVolume

```
public interface MediaStorageVolume
extends LogicalStorageVolume
```

This interface represents a media volume on a storage device and is contained within a `StorageProxy`. A MediaStorageVolume is a specialized LogicalStorageVolume that supports the recording and playback of media content through the DVR. The volume also provides a mechanism for allocating a fixed amount of storage for use by recordings on the volume.

Method Summary

void	**addFreeSpaceListener**(FreeSpaceListener listener, int level) 　　　Adds a listener that is notified when available free space is less than a specified level.
void	**allocate**(long bytes) 　　　Allocates the specified amount of storage from the containing StorageProxy for use by recordings made to this volume.
void	**allowAccess**(java.lang.String[] organizations) 　　　Adds a list of Organization strings to the set of organizations that are allowed to use this volume.
long	**getAllocatedSpace**() 　　　Gets the amount of space allocated on this volume.
java.lang.String[]	**getAllowedList**() 　　　Returns the list of Organizations who are allowed to use this volume.
long	**getFreeSpace**() 　　　Gets the remaining available space from an allocation after accounting for all used space (including recordings, time shift buffers, and meta-data).
void	**removeAccess**(java.lang.String organization) 　　　Removes an Organization from the list of Organization who are allowed to use this volume.
void	**removeFreeSpaceListener**(FreeSpaceListener listener) 　　　Removes a free space listener.

Methods inherited from interface org.ocap.storage.LogicalStorageVolume
getFileAccessPermissions, getPath, getStorageProxy, setFileAccessPermissions

Method Detail

allocate

```
void allocate(long bytes)
```

Allocates the specified amount of storage from the containing StorageProxy for use by recordings made to this volume. The volume is guaranteed to be able to use this amount of storage without requiring the deletion of the contents of other volumes and is also limited to using no more than the allocated amount of storage. The amount of space allocated may be rounded up to meet platform requirements. Once the storage on the volume reaches the amount allocated, the behavior is the same as if the storage device were full, e.g. a SpaceFullException is thrown or a RecordingAlertEvent generated.

A value of zero indicates that the volume has no minimum guaranteed size and may also use as much space as is available. Until set with the allocate() method, the space allocated is zero.

Subsequent calls to allocate() change the existing allocation. However, if a new allocation size is too small to contain existing recordings a IllegalArgumentException is thrown and the allocation size is not changed. Except when the allocation size is changed to zero which removes the limit and the guaranteed storage size. The allocated space can only be released by an explicit call to allocate() or through the deletion of the storage volume.

> **Parameters:**
> `bytes` - Number of bytes to allocate.
> **Throws:**
> `java.lang.SecurityException` - if the calling application does not have MonitorAppPermission("storage") permission.
> `java.lang.IllegalArgumentException` - if the requested amount of storage exceeds the amount available for allocation, or reduces the previous allocation making it too small for existing recordings.

getAllocatedSpace

`long` **`getAllocatedSpace`**`()`

> Gets the amount of space allocated on this volume. If the allocate method has not been called for the volume this method returns 0.
> **Returns:**
> Number of bytes allocated.

getFreeSpace

`long` **`getFreeSpace`**`()`

> Gets the remaining available space from an allocation after accounting for all used space (including recordings, time shift buffers, and meta-data). If no allocated space has been used, this method returns the same value as the getAllocatedSpace method. When this method is called on a MediaStorageVolume without an explicit allocation, as is the case when allocate has not been called or was called with a value of 0, then the value returned is the space available on the associated StorgeProxy's MEDIAFS that has not been explicitly allocated to another MediaStorageVolume.
> **Returns:**
> Number of bytes available for use from an allocation.

allowAccess

`void` **`allowAccess`**`(java.lang.String[] organizations)`

> Adds a list of Organization strings to the set of organizations that are allowed to use this volume. The volume is owned by the application that created the volume but is accessible to any record requests where the Organization string matches one of the strings in the organization array.

Note: Given Organization strings should represent an application's `organization_id`, formatted as would be found in the OrganizationName field of the application's leaf certificate. That is, the `organization_id` as a fixed length 8 character hexadecimal string (with leading zeros where required).

> **Parameters:**
> `organizations` - An array of strings representing organizations that are allowed to use this volume. The String passed as a parameter to the record method should match one of this strings to record onto this

---

volume. If an array of length 0 is passed, any application can use this volume. If null is passed in and all access to this volume has been removed by a call to the `removeAccess` method, then access is restored to the same organizations that had access before all access was removed.

Throws:

`java.lang.SecurityException` - if the calling application is not the owner of the volume and does not have MonitorAppPermission("storage") permission.

removeAccess

void **removeAccess**(java.lang.String organization)

Removes an Organization from the list of Organization who are allowed to use this volume. When application access to the volume is in progress from either the Java I/O (java.io package) or recording manager (org.ocap.dvr, org.ocap.shared.dvr packages) APIs and that application's access is removed by this method, the implementation SHALL terminate the reads or writes immediately and generate the appropriate response, e.g. IOException, interrupted recording request. This includes all forms of access from those APIs including file I/O, service recording, and recording playback.

Parameters:

`organization` - A string representing an organization that should be removed from the list of allowed organizations. Passing in null removes all application access to this volume.

Throws:

`java.lang.SecurityException` - if the calling application is not the owner and the calling application does not have MonitorAppPermission("storage") permission.

getAllowedList

java.lang.String[] **getAllowedList**()

Returns the list of Organizations who are allowed to use this volume. The volume is owned by the application that created the volume but is accessible to any record requests where the Organization string matches one of the strings in the organization array.

Returns:

An array of strings representing organizations that are allowed to use this volume. A zero length array is returned when all organizations have access. Null is returned when all access has been removed from this volume.

addFreeSpaceListener

void **addFreeSpaceListener**(FreeSpaceListener listener,
                             int level)

Adds a listener that is notified when available free space is less than a specified level. The parameter level is a percentage of the total available space in the volume. For example, a level of 10 would cause the listener to be notified when less than 10% of the volume is available for use. Determination of the level is implementation specific and the listener is notified whenever the threshold indicated by the level is crossed and available storage is less than the level parameter

Parameters:

`listener` - The listener to be added.

`level` - The level of free space remaining at which to notify the listener.

removeFreeSpaceListener

void **removeFreeSpaceListener**(FreeSpaceListener listener)

Removes a free space listener. If the parameter listener was not previously added or has already been removed this method does nothing successfully.

Parameters:

`listener` - The listener to remove.

org.ocap.dvr.storage
Interface SpaceAllocationHandler

```
public interface SpaceAllocationHandler
```

A class implementing this interface decides whether requests to allocate storage space should be allowed or not.

Method Summary

long	**allowReservation**(org.ocap.storage.LogicalStorageVolume volume, org.dvb.application.AppID app, long spaceRequested) This method should be used by the implementation to allow the SpaceAllocationHandler to grant a request to reserve space.

Method Detail

allowReservation

```
long allowReservation(org.ocap.storage.LogicalStorageVolume volume,
                      org.dvb.application.AppID app,
                      long spaceRequested)
```

This method should be used by the implementation to allow the SpaceAllocationHandler to grant a request to reserve space.

Parameters:

volume - The LogicalStorageVolume on which the reserved space is requested.

app - The requesting application.

spaceRequested - The new value of the reservation if the request is granted.

Returns:

the space granted.

Annex F   OCAP Shared DVR API (org.ocap.shared.dvr) - see [A088 Rev. 2]

Annex G   OCAP Shared DVR Navigation API (org.ocap.shared.dvr.navigation) - see [A088 Rev. 2]

Annex H　OCAP Shared Media API (org.ocap.shared.media) - see [A088 Rev. 2]

Annex I   (void)

Annex J   OCAP DVR Event API (org.ocap.dvr.event)

Package org.ocap.dvr.event

Interface Summary	
LightweightTriggerHandler	This interface represents a handler that can register for interest in specific stream types.
LightweightTriggerSession	This interface represents a session created to build an artificial carousel with a DSMCCStreamEvent.
StreamChangeListener	This interface represents a listener an application can set in order to listen for events to do with a stream of interest.

Class Summary	
LightweightTriggerManager	This class represents a manager that can be used by a privileged application to create an artificial object carousel containing a DSMCCStreamEvent in the top level.

org.ocap.dvr.event
Interface LightweightTriggerHandler

public interface **LightweightTriggerHandler**

This interface represents a handler that can register for interest in specific stream types. When notified of stream type activity of interest this handler MAY create an artificial object carousel associated with the stream and populate it with stream events that will be generated whenever the stream is played back and encounters the JMF media time attached to one of the stream events.

Method Summary

void	**notifyStreamType**(LightweightTriggerSession session) Notifies the handler when a PMT is received that contains an elementary stream with a stream type it registered interest in.

Method Detail

notifyStreamType

void **notifyStreamType**(LightweightTriggerSession session)

> Notifies the handler when a PMT is received that contains an elementary stream with a stream type it registered interest in. This SHALL include scheduled in-progress recordings happening in the background, ServiceContext presentation with time-shift enabled or disabled, and BufferingRequests. If multiple PMTs in the same transport stream contain elementary streams with stream types the handler is interested in, this method SHALL be called once for each PMT. If the same stream is being presented simultaneously this method SHALL be called for each presentation, e.g., the same service is presented in multiple service contexts simultaneously. The session parameter SHALL be started by the implementation and is ready for application use.
>
> **Parameters:**
> session - The session for the stream type of interest.

org.ocap.dvr.event
Class LightweightTriggerManager

```
java.lang.Object
  └─org.ocap.dvr.event.LightweightTriggerManager
```

public abstract class **LightweightTriggerManager**
extends java.lang.Object

This class represents a manager that can be used by a privileged application to create an artificial object carousel containing a DSMCCStreamEvent in the top level. The DSMCCStreamEvent can be populated by a privileged application. NOTE this is an expanded version of the GEM lightweight binding of the trigger API (GEM clause P.2.3.1 Lightweight binding of trigger API).

Constructor Summary

protected	**LightweightTriggerManager**() Protected constructor not callable by applications.

Method Summary

static LightweightTriggerManager	**getInstance**() Gets an instance of the manager.
abstract void	**registerHandler** (LightweightTriggerHandler handler, short streamType) Registers a handler interested in services with streams listed in the PMT with this stream type.
abstract void	**unregisterHandler** (LightweightTriggerHandler handler) Unregisters a handler that was previously registered by the registerHandler method.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LightweightTriggerManager

protected **LightweightTriggerManager**()
          Protected constructor not callable by applications.

Method Detail

getInstance

```
public static LightweightTriggerManager getInstance()
```
> Gets an instance of the manager.
> **Returns:**
> An instance of the light-weight trigger manager.

registerHandler

```
public abstract void registerHandler(LightweightTriggerHandler handler,
                                     short streamType)
```
> Registers a handler interested in services with streams listed in the PMT with this stream type. The implementation SHALL continuously monitor all available PMTs in all available transport streams for the stream type.
> **Parameters:**
> `handler` - Handler to register.
> `streamType` - a stream type as signaled in the PMT.
> **Throws:**
> `java.lang.IllegalArgumentException` - if streamType is not in the range 0x0 to 0xFF.
> `java.lang.NullPointerException` - if handler is null.
> `java.lang.SecurityException` - if the calling application is not signed.

unregisterHandler

```
public abstract void unregisterHandler(LightweightTriggerHandler handler)
```
> Unregisters a handler that was previously registered by the registerHandler method.
> **Parameters:**
> `handler` - The handle to unregister.
> **Throws:**
> `java.lang.IllegalArgumentException` - if the handler was not previously registered.

org.ocap.dvr.event
Interface LightweightTriggerSession

```
public interface LightweightTriggerSession
```

This interface represents a session created to build an artificial carousel with a DSMCCStreamEvent.

Method Summary

BufferingRequest	**getBufferingRequest**() Gets the `BufferingService` for the stream type of interest when the stream is being buffered in the background by a `BufferingService`.
OcapLocator	**getLocator**() Gets the locator for the artificial carousel.
NetworkInterface	**getNetworkInterface**() Gets the `NetworkInterface` for the stream type of interest.
int[]	**getPIDs**() Get the array of PIDs for the streams with the stream type of interest.
RecordingRequest	**getRecordingRequest**() Gets an in-progress recording request for this session.
javax.tv.service .selection.ServiceContext	**getServiceContext**() Gets the `ServiceContext` for the stream type of interest.
short	**getStreamType**() Gets the stream type this session was created for and that the handler registered interest in.
boolean	**isPresenting**() Returns an indication of service containing stream type is presenting to outputs or display, or if it is being buffered or recorded in the background.
void	**registerEvent**(java.util.Date date, java.lang.String name, int id, byte[] data) Registers a synchronized event to the stream event list for this session.
void	**setStreamChangeListener** (StreamChangeListener listener) Sets the listener for this session.
void	**stop**() Stops the session.
void	**store**() Stores the artificial carousel created for an open session with any permanent recordings made of any elementary streams in the same program as the stream type associated with the open session.

Method Detail

getLocator

OcapLocator **getLocator**()

> Gets the locator for the artificial carousel. The locator is only valid during this session. The locator returned is a valid Locator that is implementation specific and unique to the artificial carousel. The locator returned SHALL be valid and usable in a `ServiceDomain` attach method call as long as the service containing the stream is presenting or is accessible in a time-shift buffer.
>
> **Returns:**
>
> locator Locator of the artificial carousel.

getStreamType

short **getStreamType**()

> Gets the stream type this session was created for and that the handler registered interest in.
>
> **Returns:**
>
> Stream type for this session.

getPIDs

int[] **getPIDs**()

> Get the array of PIDs for the streams with the stream type of interest. The array SHALL be ordered from lowest PID number to highest. Returns the PIDs for the stream type of interest and that are associated with this session.
>
> **Returns:**
>
> The PIDs for this session.

getServiceContext

javax.tv.service.selection.ServiceContext **getServiceContext**()

> Gets the `ServiceContext` for the stream type of interest.
>
> **Returns:**
>
> If a `ServiceContext` is presenting the service containing the stream type of interest and the calling application has permission to access it, then the `ServiceContext` for this session is returned, otherwise this method returns null.

getBufferingRequest

BufferingRequest **getBufferingRequest**()

> Gets the `BufferingService` for the stream type of interest when the stream is being buffered in the background by a `BufferingService`. The return value of this method MAY be incorrect as soon as this method returns.
>
> **Returns:**
>
> If the stream type of interest is buffering in the background and is not presenting, then the `BufferingService` for the stream is returned, otherwise this method returns null.

getNetworkInterface

NetworkInterface **getNetworkInterface**()

> Gets the `NetworkInterface` for the stream type of interest.
>
> **Returns:**
>
> If a `NetworkInterface` is reserved for the stream type of interest it is returned, otherwise this method returns null.

getRecordingRequest

RecordingRequest **getRecordingRequest**()

>Gets an in-progress recording request for this session. If the stream associated with the session is being recorded this method returns the recording request for it. The return value of this method MAY be incorrect as soon as this method returns.
>
>**Returns:**
>
>RecordingRequest associated with the stream, or null if the stream is not being recorded by an in-progress RecordingRequest.

isPresenting

boolean **isPresenting**()

>Returns an indication of service containing stream type is presenting to outputs or display, or if it is being buffered or recorded in the background.
>
>**Returns:**
>
>True if the service containing stream type of interest is presenting, otherwise returns false.

registerEvent

void **registerEvent**(java.util.Date date,
                     java.lang.String name,
                     int id,
                     byte[] data)

>Registers a synchronized event to the stream event list for this session.
>
>**Parameters:**
>
>date - The time when the event is to be generated. The implementation SHALL create JMF media time from this value for use with presenting broadcast and recorded services.
>
>name - A name for the event being registered. This name SHALL appear in the list of events returned by the DSMCCStreamEvent getEventList method.
>
>id - The unique identifier of the event.
>
>data - Application specific data associated with the event. This data will be delivered with the StreamEvent when the media time is incurred in the interested stream during playback. The maximum size of this data is 4096 bytes.
>
>**Throws:**
>
>java.lang.IllegalArgumentException - if the name or id already exist, or if the data array contains more than 4096 byte entries.
>
>java.lang.IllegalStateException - if the session is not open, i.e. has been stopped by the implementation or an application.

setStreamChangeListener

void **setStreamChangeListener**(StreamChangeListener listener)

>Sets the listener for this session.
>
>**Parameters:**
>
>listener - The listener to set. If null any previously set listener is removed.

stop

void **stop**()

>Stops the session. If an artificial carousel was created and the store method was called the carousel is stored at this time. If the session was already stopped this method does nothing successfully.

store

void **store**()

> Stores the artificial carousel created for an open session with any permanent recordings made of any elementary streams in the same program as the stream type associated with the open session. The implementation SHALL adjust the media times in the stored DSMCCStreamEvent so that they occur at the same point in the recording presentation as they did in the presentation recorded from. The implementation SHALL store all events added to the DSMCCStreamEvent while the session is active, i.e. events added after the LightweightTriggerHandler.notifyStreamType(org.ocap.dvr.event.Lightweigh tTriggerSession) method was called and before the stop is called for the session. If a session does not contain any registered events that fall within the duration of a recording when the session is stopped the artificial carousel is not stored with the recording. The implementation SHALL update the stored carousel as soon as an event is registered that falls within the duration of the recording. It is illegal to call this method after a session has been closed.

> **Throws:**

> java.lang.SecurityException - if the calling application does not have file permission granted in its permission request file.

> java.lang.IllegalStateException - if this method is called after the session has been closed, or if there are no events in the artificial carousel.

org.ocap.dvr.event
Interface StreamChangeListener

public interface **StreamChangeListener**

This interface represents a listener an application can set in order to listen for events to do with a stream of interest.

Field Summary

static int	**STREAM_ACTIVITY_ENDED_REASON** Activity that caused the stream a session was opened for has stopped.
static int	**STREAM_TYPE_LOST_REASON** The stream type was lost within the transport stream.
static int	**TRANSPORT_STREAM_LOST_REASON** The transport stream a session was opened for was lost.

Method Summary

void	**notifyPIDsChanged**(int[] pids) Notifies a change in the PIDs for the service containing the stream type it is interested in.
void	**notifyPresentationChanged**(boolean presenting) Notifies the presentation status has changed.
void	**notifySessionStopped**(int reason) Notifies the handler the implementation had to stop a session.

Field Detail

TRANSPORT_STREAM_LOST_REASON

static final int **TRANSPORT_STREAM_LOST_REASON**
        The transport stream a session was opened for was lost. Most likely caused by a tune to a different
        transport stream.

STREAM_TYPE_LOST_REASON

static final int **STREAM_TYPE_LOST_REASON**
        The stream type was lost within the transport stream. Indicates the PMT changed and there is no longer a
        stream signaled with a stream type the handler is interested in.

STREAM_ACTIVITY_ENDED_REASON

static final int **STREAM_ACTIVITY_ENDED_REASON**
        Activity that caused the stream a session was opened for has stopped. This could be caused by a
        background recording being stopped due to successful completion or premature termination. If stream
        activity ends due to tune away the implementation SHALL generate the
        TRANSPORT_STREAM_LOST_REASON only. If stream activity ends for some other reason such as lack
        of storage the implementation SHALL generate this reason.

Method Detail

notifyPIDsChanged

void **notifyPIDsChanged**(int[] pids)

> Notifies a change in the PIDs for the service containing the stream type it is interested in. Changes which result in no such PIDs SHALL result in the session being stopped. The array returned contains all PIDs in the service with the stream type of interest.
>
> **Parameters:**
>
> `pids` - An array of new PID or PIDs with the stream type for which the application has registered interest. The array SHALL be ordered from the lowest PID to the highest.

notifyPresentationChanged

void **notifyPresentationChanged**(boolean presenting)

> Notifies the presentation status has changed.
>
> **Parameters:**
>
> `presenting` - When true this parameter indicates the service containing the stream type of interest is presenting to one or more outputs. When false the stream is active as a background stream with no presentation, e.g., background recording.

notifySessionStopped

void **notifySessionStopped**(int reason)

> Notifies the handler the implementation had to stop a session. The implementation SHALL stop a session if and only if one of the reasons described by the constants in this interface is encountered. If the `LightweightTriggerSession.store()` method was called for this session the artificial carousel is stored by this method.
>
> **Parameters:**
>
> `reason` - The reason the session was stopped.

Appendix I   Revision History (Informative)

I.1   ECNs included in OC-SP-OCAP-DVR-I02-050524

ECN	Date Accepted	Summary
OCAP-DVR-N-04.0647-3	8/23/04	Improving design of RecordingManager.record().
OCAP-DVR-N-04.0657-1	8/23/04	Support for JMF controls not specified when playing recorded content.
OCAP-DVR-N-04.0658-1	8/23/04	Support for recorded content playback via JMF.
OCAP-DVR-N-04.0695-2	11/2/04	Time shift options (All Java).
OCAP-DVR-N-04.0707-1	11/18/04	Recorded Applications.
OCAP-DVR-N-04.0719-1	1/6/05	AES Security Change
OCAP-DVR-N-05.0738-3	5/3/05	Establishment of a Common Core.

I.2   ECNs included in OC-SP-OCAP-DVR-I03-070508

ECN	Date Accepted	Summary
OCAP-DVR-N-05.0798-2	7/19/05	Time Shift Buffer Allocation
OCAP-DVR-N-05.0805-5	12/12/05	MHP PDR Compliance
OCAP-DVR-N-05.0806-3	10/13/05	Deletion Policy
OCAP-DVR-N-05.0817-8	5/2/06	Seamless TimeShift Buffer
OCAP-DVR-N-05.0818-3	11/2/05	Mistakes in Java API method signatures
OCAP-DVR-N-05.0819-1	10/20/05	Reset a content playback rate to 1.0 when an application that set the rate is terminated.
OCAP-DVR-N-05.0829-1	12/12/05	Identifying a RecordingRequest after deletion
OCAP-DVR-N-05.0830-2	12/12/05	Recording Priority
OCAP-DVR-N-05.0836-3	1/17/06	Remove obsolete Table 10-2; add clarifying wording
OCAP-DVR-N-05.0842-1	12/19/05	System property identifying DVR extension
OCAP-DVR-N-05.0856-2	1/31/06	Scheduled recording delay at cold boot
OCAP-DVR-N-06.0862-2	2/14/06	Don't allow rate change to prevent unexpected change of a playback state
OCAP-DVR-N-06.0863-1	2/7/06	Correction of System property identifying DVR extension
OCAP-DVR-N-06.0874-1	5/11/06	Legal states for setRecordingProperties
OCAP-DVR-N-06.0887-2	5/16/06	Recording in standby without presentation
OCAP-DVR-N-06.0891-2	5/16/06	Free Space Listener
OCAP-DVR-N-06.0892-1	5/30/06	Typo of RECORDED_SERVICE_TYPE
OCAP-DVR-N-06.0897-4	7/11/06	ServiceDetails clarification
OCAP-DVR-N-06.0898-3	9/21/06	Clarification of events during playback of recorded contents

OCAP-DVR-N-06.0903-1	6/29/06	Clarification of a return value of a record() method
OCAP-DVR-N-06.0911-10	2/8/07	Media Time Tags
OCAP-DVR-N-06.0929-3	11/16/06	Disabled MediaStorageVolume
OCAP-DVR-N-06.0931-1	9/21/06	Set Presentation
OCAP-DVR-N-06.0943-1	10/31/06	TimeShiftProperties imports
OCAP-DVR-N-07.0977-2	2/8/07	OcapRecordingProperties fix
OCAP-DVR-N-07.0994-2	4/24/07	Recording Permissions PRF Extension
OCAP-DVR-N-07.1017-1	4/24/07	MHP-PVR (A088r2) common core compliance

I.3   ECNs included in OC-SP-OCAP-DVR-I04-071220

ECN	Date Accepted	Summary
OCAP-DVR-N-07.1027-3	6/26/07	Stream_type 0xC0 for ETV support
OCAP-DVR-N-07.1048-2	8/14/07	LightweightTriggerSession method additions
OCAP-DVR-N-07.1049-1	9/18/07	Add resource priority to OcapRecordingProperties
OCAP-DVR-N-07.1057-1	7/16/07	Clarify organization name string usage
OCAP-DVR-N-07.1062-1	7/16/07	Correct ocap:monitorapplication reference
OCAP-DVR-N-07.1070-1	9/18/07	DVR SharedResourceUsage extends OCAP SharedResource Usage
OCAP-DVR-N-07.1076-2	10/2/07	Clarify recording to a detached device
OCAP-DVR-N-07.1084-1	9/25/07	Corrections regarding Lightweight triggers
OCAP-DVR-N-07.1096-1	10/16/07	Add TimeShiftProperties.removeTimeShiftListener() method
OCAP-DVR-N-07.1098-1	10/16/07	MediaStorageVolume.getFreeSpace accounts for all uses