International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.203

(11/2006)

SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

Application for Interactive Digital Television

Common core for digital video recorder platform

ITU-T Recommendation J.203

ITU-T Recommendation J.203

Common core for digital video recorder platform

Summary

This Recommendation defines the APIs, semantic guarantees and system aspects of a harmonized digital video recorder platform.

Source

ITU-T Recommendation J.203 was approved on 29 November 2006 by ITU-T Study Group 9 (2005-2008) under the ITU-T Recommendation A.8 procedure.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

		Page
1	Scope	1
2	References	1
2.1	Normatives references	1
2.2	Informative references	1
3	Definitions	3
4	Abbreviations	3
5	Conventions	4
6	General considerations	4
6.1	Purpose	4
6.2	Full conformance with this Recommendation	4
7	Recording and playback process	4
7.1	Managing scheduled recordings	4
7.2	The recording process	5
7.3	Managing completed recordings	6
7.4	Playback of scheduled recordings	6
7.5	Timeshift	7
8	Recording and playback APIs	8
8.1	Recording and recording management	8
8.2	Playback	10
8.3	Other APIs	11
8.4	Permissions	12
9	Application signalling	12
9.1	Applications recording description	12
10	Applications model	13
10.1	Application lifecycle and trick-mode playback	13
11	Security	14
11.1	Introduction (informative)	14
11.2	Permission request file	14
12	Minimum receiver requirements	14
Annex A – Application recording description		15
Annex B – Responsibilities of GEM recording specifications		17
B.1	Required	17
B.2	Optional	18
Annex C – External references; errata, clarifications and exemptions		20
C.1	Java media framework	20

Page

Annex D – API packages for digital video recorder platform common core 21
D.1 Shared digital video recorder package 21
D.2 Shared digital video recorder navigation package..... 70
D.3 Shared media package 84
Bibliography..... 98

Introduction

This Recommendation is intended to harmonize an application environment for the digital video recorder extension to [ITU-T J.202].

This work was carried out as a joint effort between DVB and CableLabs.

ITU-T Recommendation J.203

Common core for digital video recorder platform

1 Scope

This Recommendation defines a modular extension to [ITU-T J.202], and updated as GEM [ETSI TS 102 819], which defines how the recording and playback of digital video (and audio) content is integrated with the GEM [ETSI TS 102 819] platform. This Recommendation is firstly intended to be used by entities writing terminal specifications and/or standards that extend a GEM [ETSI TS 102 819] terminal specification with digital video (and audio) recording and playback. Secondly, it is intended for developers of GEM [ETSI TS 102 819] applications that wish to use digital video (and audio) recording and playback. Implementers should consult the publisher of specifications which reference GEM [ETSI TS 102 819] regarding conformance.

NOTE – This Recommendation defines the interfaces visible to applications. Application developers should not assume that any related interface is available unless it is specifically listed. Terminal standards or implementations may have other interfaces present. One of the primary goals of this Recommendation is to maximize the common aspects concerning the integration of digital video/audio recording between MHP [ETSI ES 201 812] and the various GEM [ETSI TS 102 819] terminal specifications.

2 References

2.1 Normatives references

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T J.202] ITU-T Recommendation J.202 (2005), *Harmonization of procedural content formats for interactive TV applications*.

[ETSI ES 201 812] ETSI ES 201 812 V1.1.1, Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.0.3.

[ETSI TS 102 812] ETSI TS 102 812 V1.3.1, Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.1.2.

[ETSI TS 102 819] ETSI TS 102 819, Digital Video Broadcasting (DVB); Globally Executable MHP (GEM).

2.2 Informative references

The following references are provided as informative in order not to restrict the application of this Recommendation to TVAnytime services, but users are encouraged to use the standards below where possible for the sake of harmonization.

– ETSI TS 102 822-1 V1.3.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 1: Benchmark Features.

– ETSI TS 102 822-2 V1.3.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 2: System description.

– ETSI TS 102 822-3-1 V1.3.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 3: Metadata; Sub-part 1: Phase 1 – Metadata schemas.

– ETSI TS 102 822-3-2 V1.3.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 3: Metadata; Sub-part 2: System aspects in a uni-directional environment.

– ETSI TS 102 822-3-3 V1.1.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 3: Metadata; Sub-part 3: Phase 2 – Extended Metadata Schema.

– ETSI TS 102 822-3-4 V1.1.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 3: Metadata; Sub-part 4: Phase 2 – Interstitial metadata.

– ETSI TS 102 822-4 V1.2.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 4: Content referencing.

– ETSI TS 102 822-5 V1.1.1 (2005-03), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime Phase 1"); Part 5: Rights Management and Protection (RMP) Information for Broadcast Applications.

– ETSI TS 102 822-5-1 V1.2.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 5: Rights Management and Protection (RMP) Sub-part 1: Information for Broadcast Applications.

– ETSI TS 102 822-5-2 V1.2.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 5: Rights Management and Protection (RMP) Sub-part 2: RMPI binding.

– ETSI TS 102 822-6-1 V1.3.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 6: Delivery of metadata over a bi-directional network; Sub-part 1: Service and transport.

– ETSI TS 102 822-6-2 V1.3.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 6: Delivery of metadata over a bi-directional network; Sub-part 2: Phase 1 – Service discovery.

– ETSI TS 102 822-6-3 V1.1.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 6: Delivery of metadata over a bi-directional network; Sub-part 3: Phase 2 – Exchange of Personal Profile.

– ETSI TS 102 822-7 V1.1.1 (2003-10), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime Phase 1"); Part 7: Bi-directional metadata delivery protection.

– ETSI TS 102 822-8 V1.1.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 8: Phase 2 – Interchange Data Format.

– ETSI TS 102 822-9 V1.1.1 (2006-01), Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 9: Phase 2 – Remote Programming.

3 Definitions

This Recommendation defines the following terms:

3.1 GEM recording specification: A complete specification that extends this Recommendation to create complete specification of how to integrate digital video (and audio) recording and playback with GEM terminal specifications.

3.2 GEM recording terminal: Terminal or other device that conforms to a GEM recording specification.

3.3 normal play: Play of video and/or audio content in the forward direction at a rate of 1.0. This includes live content, live content which has been time-shifted and content which is the result of a scheduled recording.

3.4 recordable streams: The streaming data within a piece of content which is to be recorded and played back. For GEM terminal specifications, these include streams identified corresponding to clauses 7.2.1 ("Audio") and 7.2.2 ("Video") of GEM [ETSI TS 102 819]. GEM recording specifications may define other types of streams to be recordable streams, for example subtitles or closed-captions.

3.5 recording: A generic term that refers to a layman's concept of a scheduled or recorded event, and does not refer to an actual Java object. The term covers recordings which have been requested but not yet started, recordings which are in progress, recordings which have completed (successfully or not) and recordings which failed with no data ever being recorded.

3.6 scheduled recordings: Recordings where the target of the recording is specified either by time, duration and channel or by an identifier that is resolved to time, duration and channel.

3.7 synthesized timeline: A timeline for a piece of content which was synthesized by the GEM recording terminal (as opposed to being included as part of the piece of content when it was transmitted).

3.8 timeline: The conceptual progress of time inherent in an item of content, which may be referred to by applications and delivered by a transmitted timeline.

3.9 timeshift recordings: Recordings where the target of the recording is currently received video and audio content.

3.10 transmitted timeline: A timeline included as part of a piece of content when that content is transmitted (e.g., DSMCC Normal Play Time).

3.11 trick mode or trick modes: A generic term for the playback of video and/or audio at speeds other than 1.0 or directions other than forwards. This includes fast and slow speed playback, both forwards and backwards, as well as pause.

3.12 trick-mode aware application: An application signalled as being aware of trick-mode playback, i.e., with the trick_mode_aware_flag set to '1'.

4 Abbreviations

This Recommendation uses the following abbreviations:

AIT Application Information Table (as defined in clause 11.4 of [ETSI ES 201 812] and [ETSI TS 102 812]).

DVR Digital Video Recorder

PDR Personal Digital Recorder

5 Conventions

The term "application" is used to cover both GEM applications and applications which are not themselves GEM applications but which are compliant with the GEM terminal specification implemented by a particular GEM recording terminal.

This Recommendation uses the terminology that something "shall not be recorded" or "should not be recorded". This is a convention for the sake of brevity and in all cases, implementations are allowed to record items that shall or should "not be recorded" and discard them during playback. The term "shall not be recorded" is simply shorter than "shall not be recorded, or if recorded, shall be discarded during playback" and the latter is what is meant.

6 General considerations

6.1 Purpose

This Recommendation is not intended, and should not be used, as a complete specification of how to integrate digital video (and audio) recording and playback with GEM terminal specifications. It is a framework upon which such a complete specification (known as a GEM recording specification) can be created.

6.2 Full conformance with this Recommendation

Table 6-1 – MHP and GEM terminal specifications

[ETSI ES 201 812]	MHP 1.0
[ETSI TS 102 812]	MHP 1.1
[ETSI TS 102 819]	GEM

Table 6-2 – MHP and GEM recording specifications

[b-ETSI TS 102 816]	PVR/PDR Extension to the Multimedia Home Platform
OpenCable Application Platform Specification [b-OC-SP-OCAP-DVR-I02-050524]	OCAP Digital Video Recorder

For avoidance of doubt, equipment which is fully conformant with this entire Recommendation apart from the above clause is not fully conformant with this Recommendation.

7 Recording and playback process

Implementations of this Recommendation shall perform the following as part of their process for recording and playback.

7.1 Managing scheduled recordings

The process for managing scheduled recordings shall include the following activities:

1) Maintaining a list of recording requests which remain in the pending state until at least the end of the validity period of the recording request;

2) Maintaining a list of recording requests that have been successfully completed, ones where recording started but failed to be successfully completed, and ones where recording was scheduled but failed to start;

3) Initiating the recording process for a pending recording request at the appropriate time, including awakening from a standby or similar power management state should this be necessary;

4) Maintaining references to recording requests that failed in the list of recording requests for at least the validity period of the recording request.

NOTE – Mechanisms for resolving conflicts between recording requests (e.g., use of the tuner) are outside the scope of this Recommendation and should be specified by GEM recording specifications.

7.2 The recording process

The recording process shall include the following activities:

1) Identifying which recordable streams should be recorded. If specific streams were specified when the recording request was originally scheduled, then those are the ones which should be recorded. If no specific streams were specified when the recording request was originally scheduled, then the default streams in the piece of content should be recorded.

   NOTE 1 – The definition of the default streams to be recorded is outside the scope of this Recommendation and should be specified by GEM recording specifications.

2) Recording the identified streams, up to the limits in the recording capability of the GEM recording terminal. Where more streams of any one type should be recorded than the GEM recording terminal can record, streams shall be prioritized according to clause 11.4 of [ETSI TS 102 819] (which in turn is according to clause 11.4.2.3 of [ETSI ES 201 812] and [ETSI TS 102 812]).

   NOTE 2 – Minimum capabilities for the number of streams of each type that GEM recording terminals must be able to record are outside the scope of this Recommendation and should be specified by GEM recording specifications.

3) Identifying recordable applications and recording them and sufficient data to reconstruct their AIT entries.

   Applications with an Application recording description where the scheduled_recording_flag is set to '0' shall not be considered as recordable.

   NOTE 3 – A more complete definition of which applications are recordable (and which are not) is outside the scope of this Recommendation and should be specified by GEM recording specifications.

   NOTE 4 – The requirements on a GEM recording terminal to monitor for dynamic data and behaviour of applications during a recording are outside the scope of this Recommendation and should be specified by GEM recording specifications.

4) Recording sufficient information about all transmitted timelines which form part of the recording in order to enable them to be accurately reconstructed during playback.

5) Generating a media time which increments linearly at a rate of 1.0 from the beginning to the end of the recording.

6) Handling the following cases relating to the interruption of power to a GEM recording terminal during a scheduled recording – the recording is in progress and power is removed – the recording starts while power is not available but power is returned before the end of the scheduled recording – the recording is in progress and power is removed and returned once or more than once – the recording starts while power is not available but power is returned and then removed once or more than once. In all these cases, as much of the content as is available while the GEM recording terminal has power shall be recorded.

7.3 Managing completed recordings

The process for managing completed recordings shall include the following activities:

1) Maintaining with all completed recordings the following information as long as the content is retained;
   - whether the recording is known to be complete or incomplete or whether this is unknown;
   - the time and channel where the recording was made;
   - the application specific data associated with the recording.
2) Deleting the recording if required (including the entry in the list of recordings, the recorded data and any other associated information) once the expiration period is past for the leaf recording request corresponding to this recording.

   NOTE – This Recommendation is silent about the precision with which the expiration period must be respected. GEM recording specifications should specify how accurately it should be enforced by implementations.

7.4 Playback of scheduled recordings

7.4.1 Process for playback

The process for playing back scheduled recordings shall include the following activities:

1) Starting the playback of recordable streams unless the initiating_replay_flag in the Application recording description is set to '1'.
2) Starting the playback of recorded applications where these form part of the recorded piece of content and their application_control_code is AUTOSTART.

   NOTE – Requirements on reconstructing the dynamic behaviour of recorded applications during playback are outside the scope of this Recommendation and should be specified by GEM recording specifications.
3) When playing content which is currently being recorded, if the end of the content to be recorded is reached and recording stops, the playback must continue without interruption (but not necessarily perfectly seamlessly), 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.
4) A time shall be synthesised which increases linearly from the start of the recorded content to the end. This shall be used as the basis of the "time base time" and "media time" as defined by JMF. No relationship is required between this time and any time that forms part of the recorded content such as MPEG PCR, STC or DSMCC NPT.
5) For all transmitted time lines which form part of the original recording, reconstruct each time line when the current media time is in the range for which that time line is valid.

7.4.2 Events during playback

During the playback of content recorded as the result of a scheduled recording, the following behaviour shall be supported:

Table 7-1 – Events during normal playback and resulting behaviour

Event	Behaviour	Result on screen
Fast forward to end of stream	End of media event generated to any registered applications	Last frame frozen
Rewind to beginning of stream	Switch to pause mode	First frame frozen
Play to end of stream	End of media generated to any registered applications	Last frame frozen

7.5 Timeshift

7.5.1 The recording process

The process for timeshift recording shall include the following activities:

1) Identifying which recordable streams are to be recorded and recording them.

   NOTE 1 – The definitions of which streams are to be recorded in timeshift recording is outside the scope of this Recommendation and should be specified by GEM recording specifications.

2) Identifying recordable applications signalled with the content being recorded at the start of the recording and recording at least those applications and sufficient data to reconstruct their AIT entries.

   NOTE 2 – The definition of which applications are recordable in timeshift (and which are not) is outside the scope of this Recommendation and should be specified by GEM recording specifications.

   NOTE 3 – The requirements for a GEM recording terminal to monitor for dynamic data and behaviour of applications during a timeshift recording are outside the scope of this Recommendation and should be specified by GEM recording specifications.

3) Identifying the transmitted timelines which form part of the content being recorded and recording sufficient information about them to enable them to be accurately reconstructed during playback.

4) Managing the timeshift buffer such that when the buffer is full, recording continues with the oldest remaining content in the buffer being progressively overwritten.

7.5.2 Playback

Playback of content recorded in timeshift mode requires a time-shift buffer to be associated with a JMF player or service context. The definition of how and when this association is made is outside the scope of this Recommendation and should be specified by GEM recording specifications.

During the playback of content recorded in timeshift mode, the behaviour shall be as defined in clause 7.4 with the following modifications:

1) The behaviour defined by Table 7-1: Events during normal playback and resulting behaviour shall be replaced by the table below.

Table 7-2 – Events during timeshift playback and resulting behaviour

Event	Behaviour
Fast forward to end of timeshift buffer (i.e., the point where newly recorded content is being written)	Switch to playback with rate = 1.0 within the buffered recording or live without destroying the contents of the timeshift buffer.
Rewind to the beginning of the timeshift buffer (i.e., the point where recorded content is being overwritten)	Switch to normal playback with rate = 1.0 within the buffered recording, from the beginning of the content in the timeshift buffer.
Play to end of timeshift buffer (i.e., bad signal reception causes a data underflow in the timeshift buffer)	Either do nothing (i.e., continue playback with rate = 1.0 within the buffered recording) or switch to live without destroying the contents of the timeshift buffer.
Pause until timeshift buffer is "full" and the content at the point where pause was made is about to be overwritten.	Switch to playback with rate = 1.0 from the point at which pause was made.

For all transmitted timelines which are valid in the timeshift buffer, reconstruct each time line when the current media time is in the range for which that time line is valid.

2) The media time for each location in the timeshift buffer shall be the value assigned to that location at the point it was recorded. There shall be no discontinuities in media time within the buffered recording including the "head" of the buffer where the content is the same as the currently received content.

3) Setting the media time to a value corresponding to POSITIVE_INFINITY shall set the playback location to the current record point, if the recording is still on-going, or, if not, to the end of the recording.

4) If the playback location is the same as the record point (i.e., the live point is being played back), the content shall be displayed with a delay of zero relative to the original broadcast content. Implementations which can display the recorded content with a delay of zero relative to live may display the recorded content. Other implementations must display the original broadcast content.

8 Recording and playback APIs

8.1 Recording and recording management

8.1.1 Overview (informative)

The recording APIs enable applications to perform recordings in two different ways:

1) Schedule a recording to be performed at some point in the future;

2) Record broadcast events in real time. This may also include the portion of the broadcast event that is in a timeshift buffer.

The implementation maintains a database of recordings, represented by the RecordingManager singleton. Applications create recordings by calling the record() method of the RecordingManager object. This operation in effect creates an entry in the recording database maintained by the implementation. These entries are represented by instances of RecordingRequest. Applications may associate application-specific data with RecordingRequests using the addAppData() method. This application-specific data could be event descriptions in an application-private format or a key into some database of information maintained by the application.

When a recording is created, applications specify a set of recording properties. One of these is an expiration period, measured from when the recording is made. As defined in clause 7.3 above, once

this expiration period has passed, the implementation will delete the recording. A second property is a validity period, measured from when the request to make the recording is made. If the recording has not been made by the time this period has passed, the recording may be deleted.

Applications may acquire a set of recordings represented by a RecordingList. Applications that wish to limit the set of recordings in a RecordingList can express this by means of a RecordingListFilter. This Recommendation defines some filters which are available to applications. GEM recording specifications may define their own filters. Applications may create their own filters. Filters can be cascaded so that the output of one filter is used as the input to a subsequent filter.

NOTE – GEM recording specifications may define rules governing the access of recordings to applications. Where this is done, the listing API will not return recordings which the calling application is not allowed to access as determined by these rules.

When data starts being recorded for a recording, a RecordedService is created. RecordedService extends the JavaTV service interface and forms the link between the recording API and the playback API.

8.1.2 Details

The org.ocap.shared.dvr and the org.ocap.shared.dvr.navigation package shall be supported.

In ServiceContextRecordingSpec, GEM recording specifications may make mandatory the otherwise optional feature of storing and recording the contents of the timeshift buffer when the startTime is in the past.

When a scheduled recording starts when the GEM recording terminal device is powered off, but then the device is powered on while the scheduled recording is still in effect the implementation shall execute the following steps:

1) Set the state of the LeafRecordingRequest in accordance with available resources.
2) If resources are available, create a RecordedService and commence an associated recording.
3) If the recording completes, set the LeafRecordingRequest state to INCOMPLETE_STATE.
4) Set the Exception to be returned by the LeafRecordingRequest.getFailedException method to org.ocap.shared.dvr.RecordingFailedException.POWER_INTERRUPTION.

When a scheduled recording is in progress and the power is removed from the GEM recording terminal, but then the device is powered on and the content to be recorded has not finished, the implementation shall execute the following steps:

1) Create a LeafRecordingRequest for the remaining recording as soon as the boot process completes and the implementation detects a scheduled recording should be in progress. Copy the application-specific data from the original LeafRecordingRequest to the newly created one. Set the state in accordance with available resources. Set the RecordingSpec to match the original recording request.
2) If resources are available, create a RecordedService and commence an associated recording.
3) If the recording completes, set the LeafRecordingRequest state to INCOMPLETE_STATE.
4) Set the Exception to be returned by the LeafRecordingRequest.getFailedException method to org.ocap.shared.dvr.RecordingFailedException.POWER_INTERRUPTION.

If power is removed and re-applied to the GEM recording terminal multiple times during the duration of a scheduled recording, a LeafRecordingRequest and RecordedService shall be created for each time the power was re-applied and resources were available to commence recording.

NOTE 1 – GEM recording specifications may add extra requirements to combine pieces of a scheduled recording interrupted by power loss into a single construct (e.g., a subclass of RecordingRequest) hence enabling them to be played back and otherwise manipulated as a single entity.

When a scheduled recording is in progress and the power is removed from the GEM recording terminal, but then the device is powered on and the content to be recorded has finished, the implementation shall execute the following steps:

1) Set the LeafRecordingRequest state to INCOMPLETE_STATE.

2) Set the Exception to be returned by the LeafRecordingRequest.getFailedException method to INSUFFICIENT_RESOURCES.

When a scheduled recording is in progress and the power is removed from the GEM recording terminal, but then the device is powered on and the content to be recorded has NOT finished, the implementation shall execute the following steps:

1) Create a LeafRecordingRequest for the remaining recording as soon as the boot process completes and the implementation detects a scheduled recording should be in progress. Copy the application-specific data from the original LeafRecordingRequest to the newly created one. Set the state in accordance with available resources. Set the RecordingSpec to match the original recording request.

2) If resources are available, create a RecordedService and commence an associated recording.

3) If the recording completes, set the LeafRecordingRequest state to INCOMPLETE_STATE.

4) Set the Exception to be returned by the LeafRecordingRequest.getFailedException method to org.ocap.shared.dvr.RecordingFailedException.POWER_INTERRUPTION.

If power is removed and re-applied to the GEM recording terminal multiple times during a scheduled recording, a LeafRecordingRequest and RecordedService shall be created for each time the power was re-applied and resources were available to commence recording.

NOTE 2 – GEM recording specifications may add extra requirements to combine pieces of a scheduled recording interrupted by power loss into a single construct (e.g., a subclass of RecordingRequest) hence enabling them to be played back and otherwise manipulated as a single entity.

8.2 Playback

8.2.1 Overview (informative)

Playback of a recording can be performed in two ways depending on whether or not it is desired to start any applications which may form part of the recording.

1) Calling the select(Service) method of a ServiceContext passing in the RecordedService to be played will select all the service components in the service including any autostart applications which have been recorded;

2) Calling the getMediaLocator() method on a RecordedService and then passing the result to javax.media.Manager.createPlayer(MediaLocator). Once a Player has been obtained, start() or syncStart() will start playback. This will only playback the recordable streams within the recording and not any applications.

Once playback has started, some of the ways of controlling it include the following:

1) Use of the JMF controls returned from Player.getControl(String) or Player.getControls() for features like audio language selection and video scaling;

2) Use of Player.setRate(float) to control the speed of playback including changing between normal speed play, fast forward, fast reverse and pause.

8.2.2 Details

The org.ocap.shared.media package shall be supported. JMF players presenting the contents of a timeshift buffer shall support the TimeshiftControl control from this package.

The following extensions shall apply to the APIs required by [ETSI TS 102 819]:

1) The method javax.tv.service.selection.ServiceContext.select(Service) shall accept instances of RecordedService as being valid inputs. When called with such an instance, the recorded content of that RecordedService shall be selected in the ServiceContext specified.

2) The method javax.media.Manager.createPlayer(MediaLocator) shall accept the mediaLocators returned by RecordedService.getMediaLocator as being valid inputs and return a JMF Player. When such a player enters the started state, the recordable streams of that RecordedService shall be presented.

3) When a RecordedService is successfully selected in a ServiceContext, the AppsDatabase shall be populated from the set of applications recorded as part of that RecordedService as if those applications were transmitted live.

   NOTE 1 – AppsDatabase should be updated to the extent that the GEM recording specification requires monitoring and reconstructing changes in the application signalling.

4) During playback of recordable streams (both playback of a scheduled recording and timeshift recording), when the rate of playback changes, a javax.media.RateChangeEvent shall be sent to all applications with ControllerListeners registered on a JMF player for that content.

5) Calls to the setRate method of a JMF Player shall control the speed of playback, including changing between normal speed play, fast forward, fast reverse and pause.

6) Calls to the method Player.setMediaTime shall attempt to start presentation of content as close as possible to the specified media time except when passed a media time returned by MediaTimeFactoryControl. SetTimeApproximations when the semantics defined by that method shall be observed.

7) When a RecordedService is being played back, calls to the method ServiceDomain.attach(..) with the Locator returned by RecordedService.getLocator shall work as specified and provide access to the broadcast file system in the recording. This Recommendation does not define requirements for access to broadcast file systems in a recording when that recording is not being played back; however, such requirements may be defined by GEM recording specifications.

When JMF players are presenting the recordable streams of a RecordedService or the contents of a timeshift buffer, at a minimum the following JMF controls (defined by [ETSI TS 102 819] or this Recommendation or as explicitly specified) shall be supported:

– org.davic.media.AudioLanguageControl;

– org.davic.media.FreezeControl;

– javax.tv.media.MediaSelectControl;

– org.ocap.shared.media.TimeLineControl;

– org.ocap.shared.media.TimeFactoryControl;

– org.davic.media.MediaTimeEventControl as defined in [b-DAVIC 1.4.1p9].

NOTE 2 – GEM recording specifications may require additional JMF controls to be supported for RecordedServices or the contents of a timeshift buffer. Different sets of JMF controls may be specified for these two cases.

8.3 Other APIs

8.3.1 Versioning

The properties listed in the following two tables shall be included in the property set of the java.lang.System class. Thus these properties can be retrieved using java.lang.System.getProperty(). Since this API returns a string, numeric return values shall be encoded as defined by java.lang.Integer.toString(int).

Table 8-1 – System properties for version interrogation

Property	Semantics	Possible values	Example
gem.recording.version.major	Major version number of the version of this Recommendation supported.	Non-negative integer value	"1"
gem.recording.version.minor	Minor version number of the version of this Recommendation supported.	Non-negative integer value	"0"
gem.recording.version.micro	Micro version number of the version of this Recommendation supported.	Non-negative integer value	"0"

8.4 Permissions

8.4.1 Unsigned applications

From RecordingPermission, only RecordingPermission("read", "own") shall be granted to unsigned applications.

8.4.2 Signed applications

When the permission request file requests the permission to create a recording request and this is granted, a RecordingPermission("create", "own") is created.

When the permission request file requests the permission to modify a recording request and this is granted, a RecordingPermission("modify", "own") is created.

When the permission request file requests the permission to delete a recording request and this is granted, a RecordingPermission("delete", "own") is created.

When the permission request file requests the permission to cancel a recording request and this is granted, a RecordingPermission("cancel", "own") is created.

GEM recording specifications may define a mechanism for permitting applications to have RecordingPermission instances whose action string is "*" but this is not required.

9 Application signalling

9.1 Applications recording description

GEM recording specifications shall define an application recording description sufficient to derive the following:

Table 9-1 – Application recording description

Function	Type
scheduled_recording__flag	boolean
Trick_mode_aware_flag	boolean
Time_shift_flag	boolean
initiating_replay_flag	boolean
label_count	unsigned integer
for(i=0;i<N0;i++){	
label_length	8-bit unsigned integer
for(j=0;i<N1;i++){	
label_payload	8-bit unsigned integer
}	
storage_properties	2-bit unsigned integer
}	

scheduled_recording_flag: This single-bit flag, when set to '1', indicates that the application is appropriate to record when the service in which it is signalled is recorded by a scheduled recording. When set to '0', it indicates that the application is inappropriate to record by a scheduled recording. Examples of why an application would be inappropriate to record include the application not having

been tested in a PDR environment or that the application is closely related to the time of transmission and would be meaningless to the end-user if played back from a recording (e.g., an application tied to a live event).

trick_mode_aware_flag: This single-bit flag, if set to '1', indicates that the application is trick-mode aware. If set to '0', the application is not aware of trick-modes.

time_shift_flag: This single-bit flag, when set to '1', indicates that the application is appropriate to record when the service in which it is signalled is recorded in time-shift recording mode. When set to '0', it indicates that the application is inappropriate to record in time-shift recording mode.

initiating_replay_flag: This single-bit flag, if set to '1', indicates that the GEM recording terminal shall not initiate the playback of the recordable streams in the same recording as the application. The application is responsible for starting this playback. If set to '0', the implementation shall initiate this playback in parallel with starting the application as would conventionally be the case.

NOTE 1 – If several applications are recorded with a program or a service and those applications are signalled as constituting the entry point of the playback program, when the related recordable streams are played back, the first related auto-start application found in the stored AIT shall be launched.

label_count: This 8-bit field identifies the number of labels that have been used.

label_length: This 8-bit field identifies the number of bytes in the label.

label_char: These 8-bit fields carry an array of bytes that label a part of the application within its transport protocol.

NOTE 2 – This Recommendation does not define which parts of applications can be labelled or the form of the label (if any). Labelling can be done at the level of files or groups of files or some lower-level construct specific to the protocol used to transport the application.

storage_properties: A field indicating the importance of storing the labelled part of the application. Values for this field are defined as follows:

0 should not be stored

1 critical to store

2 optional to store

3 reserved

GEM recording specifications that include the MHP definition of the GEM "Application Signalling" functional equivalent shall fulfil this requirement by supporting the application recording descriptor defined in Annex A.

10 Applications model

10.1 Application lifecycle and trick-mode playback

When playback of recorded content has been initiated by selecting a RecordedService, the application model and lifecycle shall be modified as follows:

1) Running applications not explicitly signalled as trick-mode aware (the trick_mode_aware_flag in the Application recording description is set to '1') shall be killed when playback changes from normal to a trick-mode. Applications explicitly signalled as trick-mode aware shall continue running.

2) When playback leaves trick-mode and returns to normal, the GEM recording terminal shall evaluate the application signalling for that point in the content and start or stop applications as necessary. Applications which are started shall be started as if the end-user had just changed to a broadcast of the content concerned.

NOTE – GEM recording specifications may permit delays in re-starting applications after the return to normal play if this is believed to improve the end-user experience, for example during repeated cycles of fast-forward/play/fast-forward/play.

Transitioning from live playback to time-shifted playback shall not automatically kill any MHP applications while the time-shifted content is being played from the end of the timeshift buffer, (i.e., the point where newly recorded content is being written).

11 Security

11.1 Introduction (informative)

This Recommendation includes two security models governing the ability of applications to operate on recording requests and completed recordings.

1) One model is based on associating attributes with MHP/OCAP applications. These attributes are expressed as Java Permission classes. Certain method calls are protected and throw a SecurityException if applications without specified Permissions call them. This model is independent of the details of the RecordingRequest concerned.

2) The second model is based on associating security attributes with individual recording requests. These attributes determine which applications may perform which operations on that recording request. This Recommendation is silent about the mechanism by which these attributes are associated with a recording request.

In the normal case, an application's ability to use the features provided by this Recommendation is governed by the intersection of both sets of attributes. Operations on a RecordingRequest will fail unless the calling application has the necessary Java Permission for the operation and the attributes associated with the RecordingRequest being operated upon permit the calling application to perform that operation. GEM recording specifications may define a mechanism for highly trusted applications to obtain a Permission which enables them to bypass the second model and have the right to perform all operations on all RecordingRequests.

11.2 Permission request file

The DTD of the permission request file defined by the GEM terminal specification shall include at least the following element and associated attributes:

```
<!ELEMENT recordingpermission EMPTY>
<!ATTLIST recordingpermission
create (true|false) "false"
modify (true|false) "false"
delete (true|false) "false"
cancel (true|false) "false"
>
```

12 Minimum receiver requirements

The following requirements shall apply to all GEM recording terminals:

1) At least the following rates of variable speed playback shall be supported: –16x, –8x, –4x, –2x, –1x, 0, 0.5x, 1x, 2x, 4x, 8x, 16x.

NOTE – For the –1x rate, it is allowed to only display i-frames.

Annex A

Application recording description

GEM recording specifications that include the MHP definition of the GEM "Application Signalling" functional equivalent shall fulfil this requirement by supporting the application recording descriptor defined as follows.

The application recording descriptor can be signalled in the application descriptor loop of the AIT. This descriptor contains extra information on application life cycle indicating in particular if an application is appropriate to use in conditions of trick-mode playback. It indicates whether this application shall or shall not be recorded, when a program, along with which this application is signalled, is recorded. It provides a means to specify the locations of data resources that shall be recorded along with the application, as well as the labels of the object carousel modules of the application that shall, should or should not be recorded.

Table A.1 – Application recording descriptor syntax

Syntax	No. of bits	Identifier	Comments / Value
application_recording_descriptor (){			
descriptor_tag	8	uimsbf	6
descriptor_length	8	uimsbf	
scheduled_recording_flag	1	bslbf	
trick_mode_aware_flag	1	bslbf	
time_shift_flag	1	bslbf	
dynamic_flag	1	bslbf	
av_synced_flag	1	bslbf	
initiating_replay_flag	1	bslbf	
reserved	7	bslbf	
label_count	8	uimsbf	N0
for(i=0;i<N0;i++){			
label_length	8	uimsbf	N1
for(j=0; j<N1; j++) {			
label_char	8	uimsbf	
}			
storage_properties	2	uimsbf	
Reserved	6		
}			
component_tag_list_length	8	uimsbf	N2
for(i=0;i<N2;i++){			
component_tag	8	uimsbf	
}			
private_length	8	uimsbf	N3
for(i=0;i<N3;i++){			
Private	8	uimsbf	
}			
for(i=0;i<N4;i++){			
reserved_future_use	8	uimsbf	
}			
}			

The semantics of the fields defined in this descriptor shall be as defined in clause 9.1 above unless otherwise specified in this clause.

descriptor_tag: This 8-bit integer with value 0x06 identifies this descriptor.

dynamic_flag: This flag indicates whether the application relies on the use of dynamic data during its execution. When set to '1', it indicates that the application relies on the presence of files (either code or data) or application signalling (e.g., application control code) which change during the lifetime of the piece of content.

NOTE 1 – This Recommendation does not define behaviour for GEM recording terminals that is conditional upon the value of this flag. GEM terminal specifications may use this flag in their determination of whether or not an application is recordable.

av_synced_flag: This flag indicates whether the application requires use of the trigger events. If set to '1', this is required.

NOTE 2 – This Recommendation does not define behaviour for GEM recording terminals that is conditional upon the value of this flag. GEM terminal specifications may use this flag in their determination of whether or not an application is recordable.

label_char: These 8-bit fields carry an array of bytes that are a module label. This label matches a label on one or more modules carried by Label descriptors in the userInfo fields of the moduleInfo structure of DIIs as defined by clause B.2.2.4.1 of [ETSI ES 201 812] and [ETSI TS 102 812].

component_tag_list_length: This integer specifies the length in number of bytes of the list of component tags.

component_tag: This field identifies a service component that delivers data that is required by the application at playback time and that shall be recorded along with the application. Components carrying recordable streams need only be listed if components other than the default should be recorded. Components that are streams carrying DSMCC object carousels or MHP application information tables should not be listed and shall be silently ignored if they are listed. Except for these, components that are streams carrying MPEG-2 private sections should be listed if their recording is desirable.

private: These bytes may be used for private extensions.

reserved_future_use: These reserved bytes may be used for future DVB extensions.

Annex B

Responsibilities of GEM recording specifications

B.1 Required

The following is a list of items identified in this Recommendation as being outside the scope of this Recommendation and which GEM recording specifications must specify.

1) Which types of stream 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";
2) Mechanisms for resolving conflicts between requested recordings (e.g., use of the tuner);
3) Minimum capabilities for the number of streams (or number of streams of each type) that a GEM recording terminal must be able to record;
4) The definition of which applications are recordable in both scheduled and timeshift recording (which need not be the same);
5) Requirements on a GEM recording terminal to monitor for dynamic data (in the DSMCC object carousel or GEM functional equivalent) during scheduled and timeshift recording (which need not be the same);
6) Requirements on a GEM recording terminal to monitor for GEM triggers or DSMCC stream events during scheduled and timeshift recording (which need not be the same);
7) Requirements on a GEM recording terminal to monitor for dynamic application signalling during scheduled and timeshift recording (which need not be the same);
8) Requirements on reconstructing the timing of dynamic data, triggers/stream events and dynamic application signalling during playback of scheduled and timeshift recordings (which need not be the same). GEM recording specifications must define requirements for both normal speed playback and trick mode playback;
9) How accurately the expiration period should be enforced by implementations;
10) The definition of at least one protocol for transmitted time lines;
11) The conditions when a JMF player or service context has a time-shift buffer attached;
12) Requirements on the size of application-specific private data which it must be possible to associate with a single key without an IllegalArgumentException being thrown;
13) Requirements on the number of entries of application-specific private data which it must be possible to associate with a single RecordingRequest without a NoMoreDataEntriesException being thrown;
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 the following table relating to "RecordingRequest specific security attributes".

Table B.1 – Methods with dependencies on recording request specific security attributes

Method
`RecordingManager.addRecordingChangedListener(RecordingListListener)`
`RecordingManager.getEntries()`
`RecordingManager.getEntries(RecordingListFilter)`
`RecordingRequest.getRecordingProperties(int)`
`RecordingRequest.add.AppData(int,java.io.Serializable)`
`RecordingRequest.removeAppData(int)`
`RecordingRequest.setRecordingProperties(RecordingSpec)`
`RecordingRequest.delete()`
`RecordingManager.record()`
`RecordingRequest.cancel()`
`RecordingRequest.stop()`
`LeafRecordingRequest.getService()`
`RecordedService.setMediaTime()`

B.2 Optional

The following is a list of items identified in this Recommendation as being outside the scope of this Recommendation and which GEM recording specifications may specify.

1) Mechanisms for controlling the extent to which one application can read or modify scheduled recordings and completed recordings made by another application;
2) Sub-classes of RecordingListFilter to filter the list of recordings in ways not supported by this Recommendation;
3) Rules governing which recordings an application can access;
4) Additional JMF controls to be supported for RecordedServices or the contents of a timeshift buffer. Different sets of JMF controls may be specified for these two cases;
5) Delays in re-starting applications after the return to normal play if this is believed to improve the end-user experience, for example when repeated cycles of fast-forward/play/fast-forward/play;
6) A mechanism to permit highly trusted applications to obtain instances of RecordingPermission whose action parameter is "*";
7) that the optional behaviour defined in the class description of ServiceContextRecordingSpec, where the contents of the time-shift buffer are stored when the startTime parameter is in the past, becomes mandatory in that particular GEM recording specification;
8) Requirements on reconstructing the timing of dynamic data, triggers/stream events and dynamic application signalling during playback of scheduled and timeshift recordings (which need not be the same). GEM recording specifications must define requirements for both normal speed playback and trick mode playback;
9) How accurately the expiration period should be enforced by implementations;
10) The definition of at least one protocol for transmitted time lines;
11) The conditions when a JMF player or service context has a time-shift buffer attached;
12) Requirements on the size of application-specific private data which it must be possible to associate with a single key without an IllegalArgumentException being thrown;

13) Requirements on the number of entries of application-specific private data which it must be possible to associate with a single RecordingRequest without a NoMoreDataEntriesException being thrown;

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 B.1 relating to "RecordingRequest specific security attributes".

Annex C

External references; errata, clarifications and exemptions

C.1 Java media framework

C.1.1 javax.media.Clock

In this Recommendation, the following text in the specification for this class shall be clarified as described below.

The transformation that a Clock defines on a TimeBase is defined by three parameters: rate, media starttime (mst), and time-base start-time (tbst). Given a time-base time (tbt), the media time (mt) can be calculated using the following transformation:

$$mt = mst + (tbt – tbst)*rate$$

The rate is simply a scale factor that is applied to the TimeBase. For example, a rate of 2.0 indicates that the Clock will run at twice the rate of its TimeBase. Similarly, a negative rate indicates that the Clock runs in the opposite direction of its TimeBase.

The time-base start-time and the media start-time define a common point in time at which the Clock and the TimeBase are synchronized.

Each successful rate change shall be considered to be "a common point in time at which the Clock and the TimeBase are synchronized" as defined above. The values of the Clock and TimeBase at the common point shall be the values at the instant of the rate change.

Annex D

API packages for digital video recorder platform common core

D.1 *Shared digital video recorder package*

D.2 *Shared digital video recorder navigation package*

D.3 *Shared media package*

D.1 Shared digital video recorder package

```
org.ocap.shared.dvr
     Package
```


D.1.1 *ServiceRecordingSpec class*

D.1.2 *AccessDeniedException class*

D.1.3 *DeletionDetails class*

D.1.4 *LeafRecordingRequest Interface*

D.1.5 *LocatorRecordingSpec Class*

D.1.6 *NoMoreDataEntriesException class*

D.1.7 *ParentRecordingRequest Interface*

D.1.8 *RecordedService Interface*

D.1.9 *RecordedServiceType Class*

D.1.10 *RecordingChangedEvent Class*

D.1.11 *RecordingChangedListener Interface*

D.1.12 *RecordingFailedException class*

D.1.13 *RecordingManager class*

D.1.14 *RecordingPermission class*

D.1.15 *RecordingProperties class*

D.1.16 *RecordingRequest interface*

D.1.17 *RecordingSpec class*

D.1.18 *RecordingTerminatedEvent class*

D.1.19 *ServiceContextRecordingSpec class*

D.1.1 ServiceRecordingSpec class

```
org.ocap.shared.dvr
     Class ServiceRecordingSpec

java.lang.Object
  +--org.ocap.shared.dvr.RecordingSpec
       +--org.ocap.shared.dvr.ServiceRecordingSpec
```

---

public class ServiceRecordingSpec

extends RecordingSpec

Specifies a recording request in terms of a Service.

When instances of this class are passed to RecordingManager.record(..), the following additional failure mode shall apply – if the end time (computed as the start time + the duration) is in the past when the record method is called, the record method shall throw an IllegalArgumentException.

When an instance of this recording spec is passed in as a parameter to the RecordingRequest.reschedule(..) method, an IllegalArgumentException is thrown if the source is different from the source specified in the current recording spec for the recording request and if the recording request is in progress state.

When instances of this class are passed to RecordingManager.record(..), if the start time is in the past and either:

– none of the content concerned is already recorded;

– some of the content concerned is already recorded but the implementation does not support including already recorded content in a scheduled recording;

then the current time shall be used as the start time and the duration reduced accordingly. This Recommendation does not require implementations to include already recorded content in scheduled recordings however GEM recording specifications may require this.

Constructor Summary
`ServiceRecordingSpec(javax.tv.service.Service source, java.util.Date startTime, long duration, RecordingProperties properties)` **Constructor**

Method Summary	
`long`	**`getDuration`**`()` Returns the duration passed as an argument to the constructor.
`javax.tv.service.Service`	**`getSource`**`()` Returns the source of the recording
`java.util.Date`	**`getStartTime`**`()` Returns the start time passed as an argument to the constructor.

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

ServiceRecordingSpec

```
public ServiceRecordingSpec(javax.tv.service.Service source,
                            java.util.Date startTime,
                            long duration,
                            RecordingProperties properties)
                     throws java.lang.IllegalArgumentException
```

Constructor.

Parameters:

`source` – the service to be recorded

`startTime` – Start time of the recording

`duration` – Length of time to record in milliseconds.

`properties` – The definition of how the recording is to be done.

Throws:

`java.lang.IllegalArgumentException` – if the source is not a broadcast service or if the duration is negative

Method Detail

getSource

```
public javax.tv.service.Service getSource()
```

Returns the source of the recording

Returns:

the source passed into the constructor

---

getStartTime

```
public java.util.Date getStartTime()
```

Returns the start time passed as an argument to the constructor.

Returns:

the start time passed into the constructor

---

getDuration

```
public long getDuration()
```

Returns the duration passed as an argument to the constructor.

Returns:

the duration passed into the constructor

D.1.2 AccessDeniedException class

```
org.ocap.shared.dvr
     Class AccessDeniedException

java.lang.Object
  +--java.lang.Throwable
       +--java.lang.Exception
            +--org.ocap.shared.dvr.AccessDeniedException
```

All Implemented Interfaces:

java.io.Serializable

---

public class **AccessDeniedException**

extends java.lang.Exception

Exception thrown when an application is blocked from operating on a RecordingRequest by security attributes associated with that RecordingRequest.

See Also:

Serialized Form

Constructor Summary
`AccessDeniedException()` Constructs an AccessDeniedException with no detail message

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString`

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

Constructor Detail

AccessDeniedException

`public` **`AccessDeniedException`**`()`

Constructs an AccessDeniedException with no detail message

D.1.3 DeletionDetails class

```
org.ocap.shared.dvr
     Class DeletionDetails

java.lang.Object
  +--org.ocap.shared.dvr.DeletionDetails
```

---

public class **DeletionDetails**

extends java.lang.Object

This class contains details about the deletion of a recorded service. Instances of this class are constructed by the implementation and returned to applications from the `getDeletionDetails` method.

Field Summary	
`static int`	**`EXPIRED`** Reason code: The recorded service was deleted by the implementation because the recording request has expired.
`static int`	**`USER_DELETED`** Reason code: The recorded service was explicitly deleted by the application.

Constructor Summary
`DeletionDetails(int reason, java.util.Date date)` Constructs a DeletionDetails

Method Summary	
java.util.Date	**getDeletionTime()** Gets the date and time when the recorded service was deleted.
int	**getReason()** Reports the reason for why the recorded service was deleted.

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

Field Detail

EXPIRED

```
public static final int EXPIRED
```

Reason code: The recorded service was deleted by the implementation because the recording request has expired.

See Also:

Constant Field Values

USER_DELETED

```
public static final int USER_DELETED
```

Reason code: The recorded service was explicitly deleted by the application.

See Also:

Constant Field Values

Constructor Detail

DeletionDetails

```
public DeletionDetails(int reason,
                       java.util.Date date)
```

Constructs a DeletionDetails

Parameters:

`reason` – the reason why the recorded service was deleted

`date` – the date and time when the recorded service was deleted

Method Detail

getReason

```
public int getReason()
```

Reports the reason for why the recorded service was deleted. This is the value as passed in to the constructor.

Returns:

the reason code for which the recorded service was deleted.

---

getDeletionTime

```
public java.util.Date getDeletionTime()
```

Gets the date and time when the recorded service was deleted. This is the value as passed in to the constructor.

Returns:

the deletion date and time.

D.1.4 LeafRecordingRequest Interface

```
org.ocap.shared.dvr
     Interface LeafRecordingRequest
```

All Superinterfaces:

RecordingRequest

---

public interface **LeafRecordingRequest**

extends RecordingRequest

This interface represents information corresponding to a leaf level recording request. The recording request represented by this interface corresponds to a recording request that has been completely resolved to a single recording.

A leaf recording request may be pending (i.e., waiting for the start-time to occur), in-progress, completed, incomplete, or failed.

While in pending state, a recording request may be in conflict for resources with other recordings. Any such conflicts must be resolved before the scheduled start time of the recording, if not, the pending recording request is expected to result in a failed recording.

Field Summary	
static int	**COMPLETED_STATE** Recording for this recording request has completed successfully.
static int	**DELETED_STATE** The recorded service corresponding to this recording request has been deleted.
static int	**FAILED_STATE** The recording request has failed.
static int	**IN_PROGRESS_INSUFFICIENT_SPACE_STATE** Recording has been initiated for this recording request and is ongoing, but the implementation has detected that storage space may not be sufficient to complete the recording.
static int	**IN_PROGRESS_STATE** Recording has been initiated for this recording request and is ongoing.
static int	**INCOMPLETE_STATE** Recording for this recording request was initiated but failed in the IN_PROGRESS_STATE before the COMPLETED_STATE could be reached.
static int	**PENDING_NO_CONFLICT_STATE** The recording request is Pending. Recording for this request is expected to complete successfully.
static int	**PENDING_WITH_CONFLICT_STATE** The recording request may not be initiated due to resource conflicts.

Method Summary	
void	**cancel()** Cancels a pending recording request.
DeletionDetails	**getDeletionDetails()** Gets detailed information about the deletion of the recorded service corresponding to this recording request.
java.lang.Exception	**getFailedException()** Gets the exception that caused the recording request to enter the FAILED_STATE, or INCOMPLETE_STATE.
RecordedService	**getService()** Returns the RecordedService corresponding to the recording request.
void	**stop()** Stops the recording for an in-progress recording request regardless of how much of the duration has been recorded.

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

PENDING_NO_CONFLICT_STATE

```
public static final int PENDING_NO_CONFLICT_STATE
```

The recording request is Pending. Recording for this request is expected to complete successfully.

See Also:

Constant Field Values

PENDING_WITH_CONFLICT_STATE

```
public static final int PENDING_WITH_CONFLICT_STATE
```

The recording request may not be initiated due to resource conflicts. The implementation has detected a resource conflict for the scheduled time of this recording request and the current resolution of the conflict does not allow this recording request to be initiated successfully.

See Also:

Constant Field Values

IN_PROGRESS_STATE

```
public static final int IN_PROGRESS_STATE
```

Recording has been initiated for this recording request and is ongoing. Recording is expected to complete successfully.

See Also:

Constant Field Values

IN_PROGRESS_INSUFFICIENT_SPACE_STATE

```
public static final int IN_PROGRESS_INSUFFICIENT_SPACE_STATE
```

Recording has been initiated for this recording request and is ongoing, but the implementation has detected that storage space may not be sufficient to complete the recording. This situation may arise when the implementation detects that the storage space may not be sufficient to complete this recording request and other recording requests that are in progress. The recording request is not expected to complete successfully. A recording request may enter this state from IN_PROGRESS_STATE or PENDING_NO_CONFLICT_STATE. Implementation may start a recording request in IN_PROGRESS_STATE based on initial estimation for space required and later change to IN_PROGRESS_INSUFFICIENT_SPACE_STATE when the implementation has enough information to compute a more accurate estimation of space needed. If an implementation cannot detect insufficient space in advance, the implementation may start the recording request in IN_PROGRESS_STATE and then transition to FAILED_STATE when space runs out. It is also possible for a recording request to start in IN_PROGRESS_INSUFFICIENT_SPACE_STATE and later move to IN_PROGRESS_STATE or COMPLETED_STATE. This could occur if other recording

requests were deleted after the start of recording or if implementation computes a better estimate of the space needed as the recording progress.

See Also:

Constant Field Values

INCOMPLETE_STATE

`public static final int` **INCOMPLETE_STATE**

Recording for this recording request was initiated but failed in the IN_PROGRESS_STATE before the COMPLETED_STATE could be reached. The RecordingRequest will contain a `RecordedService` that can be played for whatever duration was recorded.

See Also:

Constant Field Values

COMPLETED_STATE

`public static final int` **COMPLETED_STATE**

Recording for this recording request has completed successfully.

See Also:

Constant Field Values

FAILED_STATE

`public static final int` **FAILED_STATE**

The recording request has failed.

See Also:

Constant Field Values

DELETED_STATE

`public static final int` **DELETED_STATE**

The recorded service corresponding to this recording request has been deleted.

See Also:

Constant Field Values

Method Detail

cancel

```
public void cancel()
           throws java.lang.IllegalStateException,
                  AccessDeniedException
```

Cancels a pending recording request. The recording request will be deleted from the database after the successful invocation of this method. Cancelling a recording request may resolve one or more conflicts. In this case some pending recordings with conflicts would be changed to pending without conflicts.

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.

---

stop

```
public void stop()
          throws java.lang.IllegalStateException,
                 AccessDeniedException
```

Stops the recording for an in-progress recording request regardless of how much of the duration has been recorded. Moves the recording to the INCOMPLETE_STATE.

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 recording is not in the IN_PROGRESS_STATE, or IN_PROGRESS_INSUFFICIENT_SPACE_STATE.

---

getFailedException

```
public java.lang.Exception getFailedException()
                                    throws java.lang.IllegalStateException
```

Gets the exception that caused the recording request to enter the `FAILED_STATE`, or `INCOMPLETE_STATE`.

Returns:

The exception that caused the failure. The exception returned will be a RecordingFailedException.

Throws:

`java.lang.IllegalStateException` – if the recording request is not in the FAILED_STATE or INCOMPLETE_STATE.

---

getService

```
public RecordedService getService()
                          throws java.lang.IllegalStateException,
                                 AccessDeniedException
```

Returns the `RecordedService` corresponding to the recording request.

Returns:

The recorded service associated with the recording request.

Throws:

`java.lang.IllegalStateException` – if the recording request is not in INCOMPLETE_STATE, IN_PROGRESS_STATE, IN_PROGRESS_INSUFFICIENT_SPACE_STATE or COMPLETED_STATE.

`AccessDeniedException` – if the calling application is not permitted to perform this operation by RecordingRequest specific security attributes.

---

getDeletionDetails

```
public DeletionDetails getDeletionDetails()
                                  throws java.lang.IllegalStateException
```

Gets detailed information about the deletion of the recorded service corresponding to this recording request.

Returns:

The deletion details for this recording request.

Throws:

`java.lang.IllegalStateException` – if the recording request is not in the DELETED_STATE.

D.1.5 LocatorRecordingSpec Class

```
org.ocap.shared.dvr
     Class LocatorRecordingSpec

java.lang.Object
  +--org.ocap.shared.dvr.RecordingSpec
       +--org.ocap.shared.dvr.LocatorRecordingSpec
```

---

public class **LocatorRecordingSpec**

extends RecordingSpec

Specifies a recording request in terms of one or more Locators.

If multiple locators are contained within the source, all of them MUST be part of the same service.

When instances of this class are passed to RecordingManager.record(..), the following additional failure mode shall apply – if the end time (computed as the start time + the duration) is in the past when the record method is called, the record method shall throw an IllegalArgumentException.

When an instance of this recording spec is passed in as a parameter to the RecordingRequest.reschedule(..) method, an IllegalArgumentException is thrown if the source is different from the source specified in the current recording spec for the recording request and if the recording request is in progress state.

When instances of this class are passed to RecordingManager.record(..), if the start time is in the past and either:

– none of the content concerned is already recorded;

– some of the content concerned is already recorded but the implementation does not support including already recorded content in a scheduled recording;

then the current time shall be used as the start time and the duration reduced accordingly. This Recommendation does not require implementations to include already recorded content in scheduled recordings however GEM recording specifications may require this.

Constructor Summary
LocatorRecordingSpec(javax.tv.locator.Locator[] source, java.util.Date startTime, long duration, RecordingProperties properties) Constructor

Method Summary	
long	**getDuration**() Returns the duration passed as an argument to the constructor.
javax.tv.locator.Locator[]	**getSource**() Returns the source of the recording
java.util.Date	**getStartTime**() Returns the start time passed as an argument to the constructor.

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

LocatorRecordingSpec

```
public LocatorRecordingSpec(javax.tv.locator.Locator[] source,
                            java.util.Date startTime,
                            long duration,
                            RecordingProperties properties)
                     throws
javax.tv.service.selection.InvalidServiceComponentException
```

Constructor

Parameters:

`source` – Source of streams to be recorded. Implementations shall make a copy of this array before the constructor returns.

`startTime` – Start time of the recording

`duration` – Length of time to record in milliseconds

`properties` – the definition of how the recording is to be done

Throws:

`javax.tv.service.selection.InvalidServiceComponentException` – if all of the locators in the source parameter are not all in the same service.

`java.lang.IllegalArgumentException` – if duration is negative.

Method Detail

getSource

```
public javax.tv.locator.Locator[] getSource()
```

Returns the source of the recording

Returns:

the source passed into the constructor

---

getStartTime

```
public java.util.Date getStartTime()
```

Returns the start time passed as an argument to the constructor.

Returns:

the start time passed into the constructor

---

getDuration

```
public long getDuration()
```

Returns the duration passed as an argument to the constructor.

Returns:

the duration passed into the constructor

D.1.6 NoMoreDataEntriesException class

```
org.ocap.shared.dvr
     Class NoMoreDataEntriesException

java.lang.Object
  +--java.lang.Throwable
       +--java.lang.Exception
            +--org.ocap.shared.dvr.NoMoreDataEntriesException
```

All Implemented Interfaces:

java.io.Serializable

---

public class **NoMoreDataEntriesException**

extends java.lang.Exception

No more data entries allowed for this recording request.

See Also:

Serialized Form

Constructor Summary
NoMoreDataEntriesException() Constructs a NoMoreDataEntriesException with no detail message

Methods inherited from class java.lang.Throwable
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

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

Constructor Detail

NoMoreDataEntriesException

public **NoMoreDataEntriesException**()

Constructs a NoMoreDataEntriesException with no detail message

D.1.7 ParentRecordingRequest Interface

```
org.ocap.shared.dvr
     Interface ParentRecordingRequest
```

All Superinterfaces:

RecordingRequest

---

public interface **ParentRecordingRequest**

extends RecordingRequest

This interface represents information corresponding to a parent recording request. The recording request represented by this interface may have one or more child recording requests.

A parent recording request may be in the unresolved state, partially resolved state, completely resolved state, or the cancelled state.

A recording request would be in the unresolved state if the implementation does not have enough information to process this recording request. A recording request in this state may transition to partially resolved state, completely resolved state, or failed state.

A recording request would be in the partially resolved state if the implementation has enough information to schedule some, but not all, child recording requests corresponding to this recording request. This would be the case of a recording request for a series where some episodes for the series are scheduled. A recording request is in completely resolved state if all its child recordings are known and scheduled.

Field Summary	
`static int`	**`CANCELLED_STATE`** A recording request is in cancelled state, if an application has successfully called the cancel method for this recording request, but not all child recording requests have been deleted.
`static int`	**`COMPLETELY_RESOLVED_STATE`** All child recordings corresponding to this recording request have been scheduled.
`static int`	**`PARTIALLY_RESOLVED_STATE`** Some recordings corresponding to this recording request have been scheduled.
`static int`	**`UNRESOLVED_STATE`** The implementation does not have enough information to process this recording request.

Method Summary	
`void`	**`cancel`**`()` Cancels the parent recording request.
`RecordingList`	**`getKnownChildren`**`()` Gets all the child Recordings corresponding to this parent RecordingRequest.

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

UNRESOLVED_STATE

```
public static final int UNRESOLVED_STATE
```

The implementation does not have enough information to process this recording request.

See Also:

Constant Field Values

COMPLETELY_RESOLVED_STATE

```
public static final int COMPLETELY_RESOLVED_STATE
```

All child recordings corresponding to this recording request have been scheduled. A recording request is in completely resolved state, if the implementation has enough information to schedule all recordings corresponding to the recording request. A recording request in completely resolved state would have one or more child recordings.

See Also:

Constant Field Values

PARTIALLY_RESOLVED_STATE

```
public static final int PARTIALLY_RESOLVED_STATE
```

Some recordings corresponding to this recording request have been scheduled. A recording request is in partially resolved state, if the implementation has enough information to schedule some, but not all recordings corresponding to the recording request. A recording request in partially resolved state may have zero, one or more child recordings.

See Also:

Constant Field Values

CANCELLED_STATE

```
public static final int CANCELLED_STATE
```

A recording request is in cancelled state, if an application has successfully called the cancel method for this recording request, but not all child recording requests have been deleted. A recording request in this state shall be deleted by the implementation once all child recording requests have been deleted.

See Also:

Constant Field Values

Method Detail

cancel

```
public void cancel()
            throws java.lang.IllegalStateException,
                   AccessDeniedException
```

Cancels the parent recording request. The implementation shall also cancel all the child recording requests through calling their cancel() method. No more child recordings will be scheduled for this recording request or for any of its child recordings. The recording request will be deleted from the database once all child recording requests have been deleted. Cancelling a parent recording request does not delete any child recordings that cannot be cancelled (i.e., if a child recording request is not in a pending state). At the successful completion of this method, the recording request would be deleted from the database or changed state to CANCELLED_STATE.

Throws:

`java.lang.SecurityException` – if the calling application does not have RecordingPermission("cancel",..) or RecordingPermission("*",..)

`AccessDeniedException` – if the calling application is not permitted to perform this operation by RecordingRequest specific security attributes.

`java.lang.IllegalStateException` – if the state of the recording request is not in UNRESOLVED_STATE, PARTIALLY_RESOLVED_STATE or COMPLETELY_RESOLVED_STATE.

---

getKnownChildren

```
public RecordingList getKnownChildren()
                              throws java.lang.IllegalStateException
```

Gets all the child Recordings corresponding to this parent RecordingRequest. For a recording request in completely resolved state this method returns all children that are still maintained in the recording manager database (i.e., children removed from the database by calling the delete() or cancel() method will not be included in the list of child recordings). For a recording request in partially resolved state this method only returns currently known children for series.

Returns:

The list of child Recordings corresponding to this Recording; null if there are no child recording requests in the RecordingManager database.

Throws:

`java.lang.IllegalStateException` – if the recording request is not in PARTIALLY_RESOLVED_STATE or COMPLETELY_RESOLVED_STATE.

D.1.8 RecordedService Interface

```
org.ocap.shared.dvr
     Interface RecordedService
```

All Superinterfaces:

javax.tv.service.Service

---

public interface **RecordedService**

extends javax.tv.service.Service

This interface represents the recorded portion of a service that is being recorded or was recorded for a period of time. As soon as recording begins, a recorded service will be created. If the recording request fails for any reason, the recording shall be stopped normally and the recorded service shall be available containing however much of the service that was recorded.

A RecordedService has a media time attribute that determines where playback begins when the recorded service is selected on a ServiceContext. This media time is persistent and is applicable for all future service selections by all applications until the media time is changed with a call to setMediaTime.

Note the following subinterface-specific behaviour for methods defined by the javax.tv.service.Service superinterface:

– The hasMultipleInstances() method shall always return false.

– The getServiceType() method shall always return RecordedServiceType.RECORDED_SERVICE.

– The getLocator() method shall return a locator corresponding to the recorded service that is different from the locator of the originating service. This locator when passed in to the SIManager.getService(..) should return this RecordedService.

– The getName() method shall return a human readable string.

– RecordedServices shall not be included in service lists returned by the method SIManager.filterServices(..).

Method Summary	
`void`	**`delete`**`()` Deletes the recorded service.
`javax.media.Time`	**`getFirstMediaTime`**`()` Gets the JMF media time at the start of the recorded service.
`javax.media.MediaLocator`	**`getMediaLocator`**`()` Returns the MediaLocator corresponding to the RecordedService.
`javax.media.Time`	**`getMediaTime`**`()` Gets the JMF media time that was set using the method setMediaTime(..)
`long`	**`getRecordedDuration`**`()` Gets the actual duration of the content recorded.
`RecordingRequest`	**`getRecordingRequest`**`()` Gets the RecordingRequest corresponding to the RecordedService.
`java.util.Date`	**`getRecordingStartTime`**`()` Gets the time when the recording was initiated.
`void`	**`setMediaTime`**`(javax.media.Time mediaTime)` Set the JMF media time for the location from where the playback will begin when this recorded service is selected on a ServiceContext.

Methods inherited from interface javax.tv.service.Service

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

Method Detail

getRecordingRequest

```
public RecordingRequest getRecordingRequest()
```

Gets the `RecordingRequest` corresponding to the RecordedService.

Returns:

The `RecordingRequest` for the service.

---

getRecordedDuration

```
public long getRecordedDuration()
```

Gets the actual duration of the content recorded. For recordings in progress, this will return the duration of the completed part of the recording.

Returns:

The duration of the recording in milliseconds.

---

getMediaLocator

```
public javax.media.MediaLocator getMediaLocator()
```

Returns the MediaLocator corresponding to the RecordedService.

Returns:

RecordedService MediaLocator.

---

setMediaTime

```
public void setMediaTime(javax.media.Time mediaTime)
                  throws AccessDeniedException
```

Set the JMF media time for the location from where the playback will begin when this recorded service is selected on a ServiceContext.

If an instance of Time corresponding to value of 0 nanosecond, or a negative value is set, or if this method was not called for this recorded service, the playback will begin at the start of the recorded content. If the instance of Time set corresponds to positive infinity or a value more than the duration of the recorded content, the playback will begin at the live point if recording is in progress for the recorded service. If the recording is not in progress, the playback will immediately hit the end-of-media. NOTE – The media time set will be applicable for all future service selections by all applications.

Parameters:

`mediaTime` – the media time corresponding to the location from where the playback will begin when this service is selected.

Throws:

`AccessDeniedException` – if the calling application is not permitted to perform this operation by RecordingRequest specific security attributes corresponding to the RecordingRequest associated with this recorded service.

`java.lang.SecurityException` – if the calling application does not have RecordingPermission("modify",..) or RecordingPermission("*",..)

---

getMediaTime

```
public javax.media.Time getMediaTime()
```

Gets the JMF media time that was set using the method setMediaTime(..)

Returns:

the value of JMF media time that was set using the method setMediaTime(..), if that method was called on this RecordedService before. If the method setMediaTime was not called before, this method should return the JMF media time corresponding to the beginning of the RecordedService.

---

getRecordingStartTime

```
public java.util.Date getRecordingStartTime()
```

Gets the time when the recording was initiated.

Returns:

the time when the recording was initiated by the implementation.

---

delete

```
public void delete()
            throws AccessDeniedException
```

Deletes the recorded service. The method removes the recorded service and all recorded elementary streams (e.g., files and directory entries) associated with the RecordedService. The corresponding recording request will transition to DELETED_STATE.

If the recording request is in the IN_PROGRESS state the implementation will stop the recording before deleting the recorded service. If the RecordedService was being presented when it was deleted, a `PresentationTerminatedEvent` will be sent with reason SERVICE_VANISHED.

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("delete",..) or RecordingPermission("*",..)

---

getFirstMediaTime

```
public javax.media.Time getFirstMediaTime()
```

Gets the JMF media time at the start of the recorded service.

Returns:

a media time

D.1.9 RecordedServiceType Class

```
org.ocap.shared.dvr
     Class RecordedServiceType

java.lang.Object
  +--javax.tv.service.ServiceType
       +--org.ocap.shared.dvr.RecordedServiceType
```

---

public class **RecordedServiceType**

extends javax.tv.service.ServiceType

This class represents the service type value for a RecordedService.

Field Summary	
`static javax.tv.service.ServiceType`	**`RECORDED_SERVICE_TYPE`** ServiceType for a Recorded service.

Fields inherited from class javax.tv.service.ServiceType
`ANALOG_RADIO, ANALOG_TV, DATA_APPLICATION, DATA_BROADCAST, DIGITAL_RADIO, DIGITAL_TV, NVOD_REFERENCE, NVOD_TIME_SHIFTED, UNKNOWN`

Constructor Summary	
`protected`	**`RecordedServiceType`**`(java.lang.String name)` Provides an instance of RecordedServiceType.

Methods inherited from class javax.tv.service.ServiceType
`toString`

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

Field Detail

RECORDED_SERVICE_TYPE

```
public static final javax.tv.service.ServiceType RECORDED_SERVICE_TYPE
```

ServiceType for a Recorded service.

Constructor Detail

RecordedServiceType

```
protected RecordedServiceType(java.lang.String name)
```

Provides an instance of RecordedServiceType.

Parameters:

`name` – The string name of this type (i.e., "RECORDED_SERVICE").

D.1.10 RecordingChangedEvent Class

```
org.ocap.shared.dvr
     Class RecordingChangedEvent

java.lang.Object
  +--java.util.EventObject
       +--org.ocap.shared.dvr.RecordingChangedEvent
```

All Implemented Interfaces:

java.io.Serializable

---

public class **RecordingChangedEvent**

extends java.util.EventObject

Event used to notify listeners of changes in the list of recording requests maintained by the RecordingManager.

See Also:

Serialized Form

Field Summary

static int	**ENTRY_ADDED** A new `RecordingRequest` was added.
static int	**ENTRY_DELETED** A `RecordingRequest` was deleted.
static int	**ENTRY_STATE_CHANGED** The state of a `RecordingRequest` changed

Fields inherited from class java.util.EventObject

source

Constructor Summary
RecordingChangedEvent(RecordingRequest source, int newState, int oldState) Constructs the event.

Method Summary	
int	**getChange**() Returns the change to the RecordingRequest.
int	**getOldState**() Returns the old state for the RecordingRequest.
RecordingRequest	**getRecordingRequest**() Returns the RecordingRequest that caused the event.
int	**getState**() Returns the new state for the RecordingRequest.

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

ENTRY_ADDED

public static final int **ENTRY_ADDED**

A new RecordingRequest was added.

See Also:

Constant Field Values

ENTRY_DELETED

public static final int **ENTRY_DELETED**

A RecordingRequest was deleted.

See Also:

Constant Field Values

ENTRY_STATE_CHANGED

```
public static final int ENTRY_STATE_CHANGED
```

The state of a `RecordingRequest` changed

See Also:

Constant Field Values

Constructor Detail

RecordingChangedEvent

```
public RecordingChangedEvent(RecordingRequest source,
                             int newState,
                             int oldState)
```

Constructs the event.

Parameters:

`source` – the `RecordingRequest` that caused the event.

`newState` – the state the `RecordingRequest` is now in.

`oldState` – the state the `RecordingRequest` was in before the state change.

Method Detail

getRecordingRequest

```
public RecordingRequest getRecordingRequest()
```

Returns the `RecordingRequest` that caused the event.

Returns:

The `RecordingRequest` that caused the event.

---

getChange

```
public int getChange()
```

Returns the change to the `RecordingRequest`.

Returns:

whether the entry was added, deleted or modified.

---

getState

```
public int getState()
```

Returns the new state for the `RecordingRequest`.

Returns:

The new state.

---

getOldState

```
public int getOldState()
```

Returns the old state for the `RecordingRequest`.

Returns:

The old state.

D.1.11 RecordingChangedListener Interface

```
org.ocap.shared.dvr
     Interface RecordingChangedListener
```

All Superinterfaces:

java.util.EventListener

---

public interface **RecordingChangedListener**

extends java.util.EventListener

Listener to receive changes in the recording list maintained by the RecordingManager.

Method Summary	
void	**recordingChanged**(RecordingChangedEvent e) Notifies the `RecordingChangedListener` of an event generated by the `RecordingManager`.

Method Detail

recordingChanged

```
public void recordingChanged(RecordingChangedEvent e)
```

Notifies the `RecordingChangedListener` of an event generated by the RecordingManager. Events are generated when there are changes in the list of recording requests maintained by the recording manager.

Parameters:

`e` – The generated event.

D.1.12 RecordingFailedException class

```
org.ocap.shared.dvr
     Class RecordingFailedException

java.lang.Object
  +--java.lang.Throwable
       +--java.lang.Exception
            +--org.ocap.shared.dvr.RecordingFailedException
```

All Implemented Interfaces:

java.io.Serializable

---

public class **RecordingFailedException**

extends java.lang.Exception

This exception is returned when applications call the getFailedException() for a failed recording request or an incomplete recording request.

See Also:

Serialized Form

Field Summary	
static int	**ACCESS_WITHDRAWN** Reason code: Recording did not complete successfully because access to the service or some component of it were withdrawn by the system before the scheduled completion of the recording.
static int	**CA_REFUSAL** Reason code: Recording failed due to the CA system refusing to permit it.
static int	**CONTENT_NOT_FOUND** Reason code: Recording failed because the requested content could not be found in the network.
static int	**INSUFFICIENT_RESOURCES** Reason code: Recording failed due to a lack of resources required to present this service.
static int	**OUT_OF_BANDWIDTH** Reason code: Recording failed due lack of IO bandwidth to record this program.
static int	**RESOLUTION_ERROR** Reason code: The recording request failed due to errors in request resolution.
static int	**RESOURCES_REMOVED** Reason code: Recording did not complete successfully because resources needed to present the service were removed before the scheduled completion of the recording.
static int	**SERVICE_VANISHED** Reason code: Recording did not complete successfully because the service vanished from the network before the completion of the recording.
static int	**SPACE_FULL** Reason code: Recording could not complete due to lack of storage space.
static int	**TUNED_AWAY** Reason code: Recording did not complete successfully because the application selected another service on the service context.
static int	**TUNING_FAILURE** Reason code: Recording failed due to problems with tuning.
static int	**USER_STOP** Reason code: The application terminated the recording using LeafRecordingRequest.stop method or by calling the stop on the service context (if the recording spec is an instance of ServiceContextRecordingSpec).

Constructor Summary
RecordingFailedException() Constructs a RecordingFailedException with no detail message

Method Summary	
int	**getReason**() Reports the reason for which the recording request failed to complete successfully.

Methods inherited from class java.lang.Throwable
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, printStackTrace, toString

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

Field Detail

CA_REFUSAL

public static final int **CA_REFUSAL**

Reason code: Recording failed due to the CA system refusing to permit it.

See Also:

Constant Field Values

CONTENT_NOT_FOUND

public static final int **CONTENT_NOT_FOUND**

Reason code: Recording failed because the requested content could not be found in the network.

See Also:

Constant Field Values

TUNING_FAILURE

public static final int **TUNING_FAILURE**

Reason code: Recording failed due to problems with tuning.

See Also:

Constant Field Values

INSUFFICIENT_RESOURCES

`public static final int` **`INSUFFICIENT_RESOURCES`**

Reason code: Recording failed due to a lack of resources required to present this service.

See Also:

Constant Field Values

ACCESS_WITHDRAWN

`public static final int` **`ACCESS_WITHDRAWN`**

Reason code: Recording did not complete successfully because access to the service or some component of it were withdrawn by the system before the scheduled completion of the recording.

See Also:

Constant Field Values

RESOURCES_REMOVED

`public static final int` **`RESOURCES_REMOVED`**

Reason code: Recording did not complete successfully because resources needed to present the service were removed before the scheduled completion of the recording.

See Also:

Constant Field Values

SERVICE_VANISHED

`public static final int` **`SERVICE_VANISHED`**

Reason code: Recording did not complete successfully because the service vanished from the network before the completion of the recording.

See Also:

Constant Field Values

TUNED_AWAY

`public static final int` **`TUNED_AWAY`**

Reason code: Recording did not complete successfully because the application selected another service on the service context. This is applicable only if the recording spec for the recording request in an instance of ServiceContextRecordingSpec.

See Also:

Constant Field Values

USER_STOP

```
public static final int USER_STOP
```

Reason code: The application terminated the recording using LeafRecordingRequest.stop method or by calling the stop on the service context (if the recording spec is an instance of ServiceContextRecordingSpec).

See Also:

Constant Field Values

SPACE_FULL

```
public static final int SPACE_FULL
```

Reason code: Recording could not complete due to lack of storage space.

See Also:

Constant Field Values

OUT_OF_BANDWIDTH

```
public static final int OUT_OF_BANDWIDTH
```

Reason code: Recording failed due lack of IO bandwidth to record this program.

See Also:

Constant Field Values

RESOLUTION_ERROR

```
public static final int RESOLUTION_ERROR
```

Reason code: The recording request failed due to errors in request resolution.

See Also:

Constant Field Values

Constructor Detail

RecordingFailedException

```
public RecordingFailedException()
```

Constructs a RecordingFailedException with no detail message

Method Detail

getReason

```
public int getReason()
```

Reports the reason for which the recording request failed to complete successfully.

Returns:

the reason code for which the recording request failed to complete successfully.

D.1.13 RecordingManager class

```
org.ocap.shared.dvr
     Class RecordingManager

java.lang.Object
  +--org.ocap.shared.dvr.RecordingManager
```

---

public abstract class **RecordingManager**

extends java.lang.Object

RecordingManager represents the entity that performs recordings.

Constructor Summary	
protected	**RecordingManager**() Constructor for instances of this class.

Method Summary	
abstract void	**addRecordingChangedListener**(RecordingChangedListener rcl) Adds an event listener for changes in status of recording requests.
abstract RecordingList	**getEntries**() Gets the list of entries maintained by the RecordingManager.
abstract RecordingList	**getEntries**(RecordingListFilter filter) Gets the list of recording requests matching the specified filter.
static RecordingManager	**getInstance**() Gets the singleton instance of RecordingManager.
abstract RecordingRequest	**getRecordingRequest**(int id) Look up a recording request from the identifier.
abstract RecordingRequest	**record**(RecordingSpec source) Records the stream or streams according to the source parameter.
abstract void	**removeRecordingChangedListener**(RecordingChangedListener rcl) Removes a registered event listener for changes in status of recording requests.

Methods inherited from class java.lang.Object

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

Constructor Detail

RecordingManager

`protected` **`RecordingManager`**`()`

Constructor for instances of this class. This constructor is provided for the use of implementations and specifications which extend this specification. Applications shall not define sub-classes of this class. Implementations are not required to behave correctly if any such application defined sub-classes are used.

Method Detail

getEntries

`public abstract RecordingList` **`getEntries`**`()`

Gets the list of entries maintained by the RecordingManager. This list includes both parent and leaf recording requests. For applications with RecordingPermission("read", "own"), only RecordingRequests of which the calling application has visibility as defined by any RecordingRequest specific security attributes will be returned. For applications with RecordingPermission("read", "*"), all RecordingRequests will be returned.

Returns:

an instance of RecordingList

Throws:

`java.lang.SecurityException` – if the calling application does not have RecordingPermission("read",..) or RecordingPermission("*",..)

---

getEntries

```
public abstract RecordingList getEntries(RecordingListFilter filter)
```

Gets the list of recording requests matching the specified filter. For applications with RecordingPermission("read", "own"), only RecordingRequests of which the calling application has visibility as defined by any RecordingRequest specific security attributes will be returned. For applications with RecordingPermission("read", "*"), all RecordingRequests matching the specified filter will be returned.

Parameters:

`filter` – the filter to use on the total set of recording requests

Returns:

an instance of RecordingList

Throws:

`java.lang.SecurityException` – if the calling application does not have RecordingPermission("read",..) or RecordingPermission("*",..)

---

addRecordingChangedListener

```
public abstract void addRecordingChangedListener(RecordingChangedListener rcl)
```

Adds an event listener for changes in status of recording requests. For applications with RecordingPermission("read", "own"), the listener parameter will only be informed of changes that affect RecordingRequests of which the calling application has visibility as defined by any RecordingRequest specific security attributes. For applications with RecordingPermission("read", "*"), the listener parameter will be informed of all changes.

Parameters:

`rcl` – The listener to be registered.

Throws:

`java.lang.SecurityException` – if the calling application does not have RecordingPermission("read",..) or RecordingPermission("*",..)

---

removeRecordingChangedListener

```
public abstract void
removeRecordingChangedListener(RecordingChangedListener rcl)
```

Removes a registered event listener for changes in status of recording requests. If the listener specified is not registered then this method has no effect.

Parameters:

`rcl` – the listener to be removed.

---

record

```
public abstract RecordingRequest record(RecordingSpec source)
                                 throws java.lang.IllegalArgumentException,
                                        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.

Parameters:

`source` – specification of stream or streams to be recorded and how they are to be recorded.

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

`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("*",..)

---

getInstance

```
public static RecordingManager getInstance()
```

Gets the singleton instance of RecordingManager.

Returns:

an instance of RecordingManager

---

getRecordingRequest

```
public abstract RecordingRequest getRecordingRequest(int id)
                                          throws java.lang.IllegalArgumentException
```

Look up a recording request from the identifier. Implementations of this method should be optimized considering the likely very large number of recording requests. For applications with RecordingPermission("read", "own"), only RecordingRequests of which the calling application has visibility as defined by any RecordingRequest specific security attributes will be returned.

Parameters:

`id` – an identifier as returned by RecordingRequest.getId

Returns:

the corresponding RecordingRequest

Throws:

`java.lang.IllegalArgumentException` – if there is no recording request corresponding to this identifier or if the recording request is not visible as defined by RecordingRequest specific security attributes

`java.lang.SecurityException` – if the calling application does not have RecordingPermission("read",..) or RecordingPermission("*",..)

See Also:

`RecordingRequest.getId()`

D.1.14 RecordingPermission class

```
org.ocap.shared.dvr
     Class RecordingPermission

java.lang.Object
  +--java.security.Permission
       +--org.ocap.shared.dvr.RecordingPermission
```

All Implemented Interfaces:

java.security.Guard, java.io.Serializable

---

public final class **RecordingPermission**

extends java.security.Permission

Controls access to recording features by an application. The name can be one of the values shown in the following list:

– "create" – schedule a RecordingRequest
– "read" – obtain the list of RecordingRequests
– "modify" – modify properties or application specific data for a RecordingRequest
– "delete" – delete a RecordingRequest including recorded content
– "cancel" – cancel a pending RecordingRequest
– "*" – all of the above

The action can be "own" and "*". The action "own" is intended for use by normal applications. The action "*" is intended for use only by specially privileged applications and permits the operation defined by the name to be applied to all RecordingRequests regardless of any per-application restrictions associated with the RecordingRequest.

Granting of this permission shall include granting access to any storage devices required for the operations specified in the name parameter. No additional low permissions (e.g., FilePermission) are subsequently needed.

See Also:

Serialized Form

Constructor Summary

RecordingPermission(java.lang.String name, java.lang.String action) Creates a new RecordingPermission with the specified name and action.

Method Summary	
boolean	**equals**(java.lang.Object obj) Checks two RecordingPermission objects for equality
java.lang.String	**getActions**() Returns the actions as passed into the constructor.
int	**hashCode**() Returns the hash code value for this object.
boolean	**implies**(java.security.Permission p) Checks if this RecordingPermission "implies" the specified Permission.

Methods inherited from class java.security.Permission
checkGuard, getName, newPermissionCollection, toString

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

Constructor Detail

RecordingPermission

```
public RecordingPermission(java.lang.String name,
                           java.lang.String action)
```

Creates a new RecordingPermission with the specified name and action.

Parameters:

`name` – "create", "read", "modify", "delete", "cancel" or "*"

`action` – "own" or "*"

Method Detail

implies

```
public boolean implies(java.security.Permission p)
```

Checks if this RecordingPermission "implies" the specified Permission.

Parameters:

`p` – the permission to check against

Returns:

true if the specified permission is implied by this object, false if not.

equals

```
public boolean equals(java.lang.Object obj)
```

Checks two RecordingPermission objects for equality

Parameters:

`obj` – the object to test for equality with this object.

Returns:

true if obj is a RecordingPermission with the same name and action as this RecordingPermission object

---

hashCode

```
public int hashCode()
```

Returns the hash code value for this object.

Returns:

a hash code value for this object.

---

getActions

```
public java.lang.String getActions()
```

Returns the actions as passed into the constructor.

Returns:

the actions as a String

D.1.15 RecordingProperties class

```
org.ocap.shared.dvr
     Class RecordingProperties

java.lang.Object
  +--org.ocap.shared.dvr.RecordingProperties
```

---

public abstract class **RecordingProperties**

extends java.lang.Object

Base class for specifying properties defining how a recording is to be made.

Constructor Summary
RecordingProperties(long expirationPeriod) Constructor

Method Summary	
long	**getExpirationPeriod**() Returns the value of the expiration period.

Methods inherited from class java.lang.Object

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

Constructor Detail

RecordingProperties

`public` **`RecordingProperties`**`(long expirationPeriod)`

Constructor

Parameters:

`expirationPeriod` – The period in seconds after the initiation of recording when leaf recording requests with this recording property are deemed as expired.

Method Detail

getExpirationPeriod

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

Returns the value of the expiration period.

Returns:

The expiration period as passed into the constructor.

D.1.16 RecordingRequest interface

```
org.ocap.shared.dvr
     Interface RecordingRequest
```

All Known Subinterfaces:

LeafRecordingRequest, ParentRecordingRequest

---

public interface **RecordingRequest**

This interface represents information corresponding to a recording request. The recording request represented by this interface may correspond to a single recording request or a series of other recording requests. Recording requests are hierarchical in nature. Implementations may resolve a recording request to a single recording request or to a series of other recording requests each of which may get further resolved into a single recording request or a series of recording requests. For example, a recording request for "Sex and the City" may resolve to multiple recording requests, each for a particular season of the show. Each of these recording requests may get further resolved into multiple recording requests, each for a single episode. The implementation creates a recording request in response to the RecordingManager.record(RecordingSpec) method. The implementation also creates recording requests when a recording request is further resolved.

A recording request may either be a parent recording request or a leaf recording request. States for a recording request are defined in `ParentRecordingRequest` and `LeafRecordingRequest`. A recording request may be in any of the states corresponding to a leaf recording request or a parent recording request.

Method Summary	
void	**addAppData**(java.lang.String key, java.io.Serializable data) Adds application specific private data.
void	**delete**() Deletes the recording request from the database.
java.io.Serializable	**getAppData**(java.lang.String key) Gets application data corresponding to specified key.
org.dvb.application.AppID	**getAppID**() Gets the application identifier of the application that owns this recording request.
int	**getId**() Returns an identifier for this recording request.
java.lang.String[]	**getKeys**() Gets all Application specific data associated with this recording request.
RecordingRequest	**getParent**() Gets the parent recording request corresponding to this recording request.
RecordingSpec	**getRecordingSpec**() Returns the RecordingSpec corresponding to the recording request.
RecordingRequest	**getRoot**() Gets the root recording request corresponding to this recording request.
int	**getState**() Returns the state of the recording request.
boolean	**isRoot**() Checks whether the recording request was a root recording request generated when the application called the RecordingManager.record(..)
void	**removeAppData**(java.lang.String key) Removes Application specific private data corresponding to the specified key.
void	**reschedule**(RecordingSpec newRecordingSpec) Modifies the details of a recording request.
void	**setRecordingProperties**(RecordingProperties properties) Modifies the RecordingProperties corresponding to the RecordingSpec for this recording request.

Method Detail

getState

```
public int getState()
```

Returns the state of the recording request.

Returns:

State of the recording request.

---

isRoot

```
public boolean isRoot()
```

Checks whether the recording request was a root recording request generated when the application called the RecordingManager.record(..) method. The implementation should create a root recording request corresponding to each successful call to the record method.

Returns:

True, if the recording request is a root recording request, false if the recording request was generated during the process of resolving another recording request.

---

getRoot

```
public RecordingRequest getRoot()
```

Gets the root recording request corresponding to this recording request. A root recording request is the recording request that was returned when the application called the RecordingManager.record(..) method.

If the current recording request is a root recording request, the current recording request is returned.

Returns:

the root recording request for this recording request, null if the application does not have read access permission for the root recording request.

---

getParent

```
public RecordingRequest getParent()
```

Gets the parent recording request corresponding to this recording request.

Returns:

the parent recording request for this recording request, null if the application does not have read access permission for the parent recording request or if this recording request is the root recording request.

---

getRecordingSpec

```
public RecordingSpec getRecordingSpec()
```

Returns the RecordingSpec corresponding to the recording request. This will be either the source as specified in the call to the record(..) method which caused this recording request to be created or the RecordingSpec generated by the system during the resolution of the original application specified RecordingSpec. Any modification to the RecordingSpec due to any later calls to the SetRecordingProperties methods on this instance will be reflected on the returned RecordingSpec.

When the implementation generates a recording request while resolving another recording request, a new instance of the RecordingSpec is created with an identical copy of the RecordingProperties of the parent recording request.

Returns:

a RecordingSpec containing information about this recording request.

---

setRecordingProperties

```
public void setRecordingProperties(RecordingProperties properties)
                           throws java.lang.IllegalStateException,
                                  AccessDeniedException
```

Modifies the RecordingProperties corresponding to the RecordingSpec for this recording request. Applications may change any properties associated with a recording request by calling this method. Changing the properties may result in changes in the states of this recording request. Changing the properties of a parent recording request will not automatically change the properties of any of its child recording requests that are already created. Any child recording requests created after the invocation of this method will inherit the new values for the properties.

Parameters:

`properties` – the new recording properties to set.

Throws:

`java.lang.IllegalStateException` – if changing one of the parameters that have been modified in the new recording properties is not legal for the current state of the recording request.

`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("modify",..) or RecordingPermission("*",..)

---

delete

```
public void delete()
            throws AccessDeniedException
```

Deletes the recording request from the database. The method removes the recording request, all its descendant recording requests, as well as the corresponding `RecordedService` objects and all recorded elementary streams (e.g., files and directory entries) associated with the RecordedService. If any application calls any method on stale references of removed objects the implementation shall throw an IllegalStateException.

If the recording request is in the IN_PROGRESS state the implementation will stop the recording before deleting the recording request. If a RecordedService was being presented when it was deleted, a `PresentationTerminatedEvent` will be sent with reason SERVICE_VANISHED.

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("delete",..) or RecordingPermission("*",..)

---

addAppData

```
public void addAppData(java.lang.String key,
                       java.io.Serializable data)
                throws NoMoreDataEntriesException,
                       AccessDeniedException
```

Adds application-specific private data. If the key is already in use, the data corresponding to key is overwritten.

Parameters:

`key` – the ID under which the data is to be added

`data` – the data to be added

Throws:

`java.lang.SecurityException` – if the calling application does not have RecordingPermission("modify",..) or RecordingPermission("*",..)

`java.lang.IllegalArgumentException` – if the size of the data is more than the size supported by the implementation within the constraints of the GEM recording specification

`NoMoreDataEntriesException` – if storing this data exceeds the number of entries supported by the implementation within the constraints of the GEM recording specification

`AccessDeniedException` – if the calling application is not permitted to perform this operation by RecordingRequest specific security attributes.

---

getAppID

```
public org.dvb.application.AppID getAppID()
```

Gets the application identifier of the application that owns this recording request. The owner of a root recording request is the application that called the RecordingManager.record(..) method. The owner of a non-root recording request is the owner of the root for the recording request.

Returns:

Application identifier of the owning application.

---

getKeys

```
public java.lang.String[] getKeys()
```

Gets all Application-specific data associated with this recording request.

Returns:

All keys corresponding to the RecordingRequest; Null if there is no application data.

---

getAppData

```
public java.io.Serializable getAppData(java.lang.String key)
```

Gets application data corresponding to specified key.

Parameters:

`key` – the key under which any data is to be returned

Returns:

the application data corresponding to the specified key; Null if there is no data corresponding to the specified key.

---

removeAppData

```
public void removeAppData(java.lang.String key)
                  throws AccessDeniedException
```

Remove Application-specific private data corresponding to the specified key. This method exits silently if there was no data corresponding to the key.

Parameters:

`key` – the key under which data is to be removed

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("modify",..) or RecordingPermission("*",..)

---

reschedule

```
public void reschedule(RecordingSpec newRecordingSpec)
                throws AccessDeniedException
```

Modifies the details of a recording request. The recording request shall be re-evaluated based on the newly provided RecordingSpec. Rescheduling a root recording request may result in state transitions for the root recording request or its child recording requests. Rescheduling a root recording request may also result in the scheduling of one or more new child recording requests, or the deletion of one or more pending child recording requests.

NOTE – If the recording request or one of its child recording request is in IN_PROGRESS_STATE or IN_PROGRESS_INSUFFICIENT_SPACE_STATE, any changes to the start time shall be ignored. In this case all other valid parameters are applied. If the new value for a parameter is not valid (e.g., the start-time and the duration is in the past), the implementation shall ignore that parameter. In-progress recordings shall continue uninterrupted, if the new recording spec does not request the recording to be stopped.

Parameters:

`newRecordingSpec` – the new recording spec that shall be used to reschedule the root RecordingRequest.

Throws:

`java.lang.IllegalArgumentException` – if the new recording spec and the current recording spec for the recording request are different sub-classes of RecordingSpec.

`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("modify",..) or RecordingPermission("*",..)*

---

getId

`public int` **`getId`**`()`

Returns an identifier for this recording request. The identifier shall uniquely identify this recording request among all others in the GEM recording terminal. The identifier shall be permanently associated with this recording request as long as this recording request remains in the GEM recording terminal and in particular shall survive power to the GEM recording terminal being interrupted. This is to enable applications to store these IDs in persistent storage for later retrieval by another application or another instance of the same application.

Since identifiers may be held in persistent storage by applications, implementations should not re-use identifiers of recording requests which are no longer held in the GEM recording terminal as this would confuse applications which still have references to those recording requests in their persistent storage.

Returns:

an identifier

See Also:

`RecordingManager.getRecordingRequest(int)`

D.1.17 RecordingSpec class

```
org.ocap.shared.dvr
     Class RecordingSpec

java.lang.Object
  +--org.ocap.shared.dvr.RecordingSpec
```

Direct Known Subclasses:

LocatorRecordingSpec, ServiceContextRecordingSpec, ServiceRecordingSpec

---

public abstract class **RecordingSpec**

extends java.lang.Object

Base class for specifying what to record and how to record it.

Constructor Summary
`RecordingSpec``(RecordingProperties properties)` Constructor.

Method Summary	
RecordingProperties	**getProperties**() Returns the description of how the recording is to be done.

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

Constructor Detail

RecordingSpec

```
public RecordingSpec(RecordingProperties properties)
```

Constructor.

Parameters:

properties – The definition of how the recording is to be done.

Method Detail

getProperties

```
public RecordingProperties getProperties()
```

Returns the description of how the recording is to be done.

Returns:

The properties to use for the recording.

D.1.18 RecordingTerminatedEvent class

```
org.ocap.shared.dvr
     Class RecordingTerminatedEvent

java.lang.Object
  +--java.util.EventObject
       +--javax.tv.service.selection.ServiceContextEvent
            +--org.ocap.shared.dvr.RecordingTerminatedEvent
```

All Implemented Interfaces:

java.io.Serializable

---

public class **RecordingTerminatedEvent**

extends javax.tv.service.selection.ServiceContextEvent

An Event Notifying that recording has terminated for the ServiceContext. This event is generated by a ServiceContext that is presenting a time-shifted service or a service that is being recorded. The presentation is not yet terminated as the playback point is time-delayed. This event is generated only when the playback point is not the same as the live point. A PresentationTerminatedEvent will be generated when the playback point catches up with the point of record termination.

See Also:

Serialized Form

Field Summary	
static int	**ACCESS_WITHDRAWN** Reason code: Access to the service or some component of it has been withdrawn by the system.
static int	**RESOURCES_REMOVED** Reason code: Resources needed to record the service have been removed.
static int	**SCHEDULED_STOP** Reason code: The recording was terminated normally as scheduled.
static int	**SERVICE_VANISHED** Reason code: The service vanished from the network.
static int	**USER_STOP** Reason code: The user requested that the recording be stopped.

Fields inherited from class java.util.EventObject
source

Constructor Summary
RecordingTerminatedEvent(javax.tv.service.selection.ServiceContext source, int reason) Constructs the event.

Method Summary	
int	**getReason**() Returns the reason for the record termination.

Methods inherited from class javax.tv.service.selection.ServiceContextEvent
getServiceContext

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

SERVICE_VANISHED

```
public static final int SERVICE_VANISHED
```

Reason code: The service vanished from the network.

See Also:

Constant Field Values

RESOURCES_REMOVED

```
public static final int RESOURCES_REMOVED
```

Reason code: Resources needed to record the service have been removed. Will be generated if the `ServiceContext` stop method is called.

See Also:

Constant Field Values

ACCESS_WITHDRAWN

```
public static final int ACCESS_WITHDRAWN
```

Reason code: Access to the service or some component of it has been withdrawn by the system. An example of this is the end of a free preview period for IPPV content.

See Also:

Constant Field Values

SCHEDULED_STOP

```
public static final int SCHEDULED_STOP
```

Reason code: The recording was terminated normally as scheduled.

See Also:

Constant Field Values

USER_STOP

```
public static final int USER_STOP
```

Reason code: The user requested that the recording be stopped. Also, if the `RecordingRequest` stop method is called.

See Also:

Constant Field Values

Constructor Detail

RecordingTerminatedEvent

```
public
RecordingTerminatedEvent(javax.tv.service.selection.ServiceContext source,
                         int reason)
```

Constructs the event.

Parameters:

`source` – The `ServiceContext` that generated the event.

Method Detail

getReason

```
public int getReason()
```

Returns the reason for the record termination.

Returns:

Termination reason; see constants in this class.

D.1.19 ServiceContextRecordingSpec class

```
org.ocap.shared.dvr
     Class ServiceContextRecordingSpec

java.lang.Object
  +--org.ocap.shared.dvr.RecordingSpec
       +--org.ocap.shared.dvr.ServiceContextRecordingSpec
```

---

public class **ServiceContextRecordingSpec**

extends RecordingSpec

Specifies a recording request in terms of what is being presented on a ServiceContext. The streams that are being presented in the indicated ServiceContext parameter are recorded. If the Service being recorded from is tuned away, recording SHALL be terminated. If the startTime is in the past and the source javax.tv.service.selection.ServiceContext is associated with a timeshift buffer, the contents of the timeshift buffer may be immediately stored to the destination beginning at the startTime, if possible, up to the live broadcast point. If the timeshift buffer does not contain the source from the startTime, as much of the source may be recorded as possible. If the startTime is in the past, but a timeshift buffer cannot be associated with the recording, the recording begins from the live broadcast point. From there, the contents of the live broadcast are recorded until the remaining duration is satisfied.

When instances of this class are passed to RecordingManager.record(..), the following additional failure modes shall apply:

– IllegalArgumentException SHALL be thrown if serviceContext is not presenting a broadcast service or if the startTime is in the future;

– SecurityException SHALL be thrown if the application does not have permission to access the service context.

When an instance of this recording spec is passed in as a parameter to the RecordingRequest.reschedule(..) method, an IllegalArgumentException is thrown if the service context parameter is different from the service context specified in the current recording spec for the recording request.

Constructor Summary
ServiceContextRecordingSpec(javax.tv.service.selection.ServiceContext serviceContext, java.util.Date startTime, long duration, RecordingProperties properties) Constructor

Method Summary	
long	**getDuration**() Returns the duration passed as an argument to the constructor.
javax.tv.service.selection.ServiceContext	**getServiceContext**() Returns the ServiceContext to record from
java.util.Date	**getStartTime**() Returns the start time passed as an argument to the constructor.

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

ServiceContextRecordingSpec

```
public
ServiceContextRecordingSpec(javax.tv.service.selection.ServiceContext serviceCon
text,
                            java.util.Date startTime,
                            long duration,
                            RecordingProperties properties)
                     throws java.lang.IllegalArgumentException
```

Constructor

Parameters:

`serviceContext` – The ServiceContext to record from.

`startTime` – Start time of the recording. If the start time is in the future when the RecordingManager.record(..) method is called with this ServiceContextRecordingSpec as an argument, the record method will throw an IllegalArgumentException.

`duration` – Length of time to record in milliseconds.

`properties` – the definition of how the recording is to be done

Throws:

`java.lang.IllegalArgumentException` – if duration is negative

Method Detail

getServiceContext

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

Returns the ServiceContext to record from

Returns:

the ServiceContext instance passed into the constructor

---

getStartTime

`public java.util.Date` **`getStartTime`**`()`

Returns the start time passed as an argument to the constructor.

Returns:

the start time passed into the constructor

---

getDuration

`public long` **`getDuration`**`()`

Returns the duration passed as an argument to the constructor.

Returns:

the duration passed into the constructor

D.2 Shared digital video recorder navigation package

```
org.ocap.shared.dvr.navigation
     Package
```


D.2.1 RecordingStateFilter class
D.2.2 AppIDFilter class
D.2.3 OrgIDFilter class
D.2.4 RecordingList interface
D.2.5 RecordingListComparator interface
D.2.6 RecordingListFilter class
D.2.7 RecordingListIterator interface

D.2.1 RecordingStateFilter class

```
org.ocap.shared.dvr.navigation
     Class RecordingStateFilter

java.lang.Object
  +--org.ocap.shared.dvr.navigation.RecordingListFilter
       +--org.ocap.shared.dvr.navigation.RecordingStateFilter
```

---

public class **RecordingStateFilter**

extends RecordingListFilter

Filter to filter based on values returned by the getState method in RecordingRequest.

Constructor Summary
RecordingStateFilter(int state) Constructs the filter based on a particular state type (PENDING, FAILED, etc.).

Method Summary	
boolean	**accept**(RecordingRequest entry) Tests if the given RecordingRequest passes the filter.
int	**getFilterValue**() Reports the value of state used to create this filter.

Methods inherited from class org.ocap.shared.dvr.navigation.RecordingListFilter
setCascadingFilter

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

Constructor Detail

RecordingStateFilter

`public RecordingStateFilter(int state)`

Constructs the filter based on a particular state type (PENDING, FAILED, etc.).

Parameters:

`state` – Value for matching the state of a `RecordingRequest` instance.

Method Detail

getFilterValue

`public int getFilterValue()`

Reports the value of state used to create this filter.

Returns:

The value of state used to create this filter.

---

accept

`public boolean accept(RecordingRequest entry)`

Tests if the given `RecordingRequest` passes the filter.

Specified by:

`accept` in class `RecordingListFilter`

Parameters:

`entry` – An individual RecordingRequest to be evaluated against the filtering algorithm.

Returns:

`true` if `RecordingRequest` contained within the RecordingRequest parameter is in the state indicated by the filter value; `false` otherwise.

D.2.2 AppIDFilter class

```
org.ocap.shared.dvr.navigation
     Class AppIDFilter

java.lang.Object
  +--org.ocap.shared.dvr.navigation.RecordingListFilter
       +--org.ocap.shared.dvr.navigation.AppIDFilter
```

---

public class **AppIDFilter**

extends RecordingListFilter

Filter to filter based on AppID.

Constructor Summary
AppIDFilter(org.dvb.application.AppID appID) Constructs the filter based on a particular AppID.

Method Summary	
boolean	**accept**(RecordingRequest entry) Tests if the given RecordingRequest passes the filter.
org.dvb.application.AppID	**getFilterValue**() Reports the value of AppID used to create this filter.

Methods inherited from class org.ocap.shared.dvr.navigation.RecordingListFilter
setCascadingFilter

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

Constructor Detail

AppIDFilter

public **AppIDFilter**(org.dvb.application.AppID appID)

Constructs the filter based on a particular AppID.

Parameters:

appID – AppID value for matching RecordingRequests.

Method Detail

getFilterValue

public org.dvb.application.AppID **getFilterValue**()

Reports the value of AppID used to create this filter.

Returns:

The value of AppID used to create this filter.

---

accept

public boolean **accept**(RecordingRequest entry)

Tests if the given RecordingRequest passes the filter.

Specified by:

accept in class RecordingListFilter

Parameters:

`entry` – An individual `RecordingRequest` to be evaluated against the filtering algorithm.

Returns:

`true` if `RecordingRequest` is of the type indicated by the filter value; `false` otherwise.

D.2.3 OrgIDFilter class

```
org.ocap.shared.dvr.navigation
     Class OrgIDFilter

java.lang.Object
  +--org.ocap.shared.dvr.navigation.RecordingListFilter
       +--org.ocap.shared.dvr.navigation.OrgIDFilter
```

---

public class **OrgIDFilter**

extends RecordingListFilter

Filter to filter based on OrgID.

Constructor Summary
`OrgIDFilter``(int orgID)` Constructs the filter based on a particular organization ID.

Method Summary	
`boolean`	**`accept`**`(RecordingRequest entry)` Tests if the given `RecordingRequest` passes the filter.
`int`	**`getFilterValue`**`()` Reports the value of the organization ID used to create this filter.

Methods inherited from class org.ocap.shared.dvr.navigation.RecordingListFilter
`setCascadingFilter`

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

Constructor Detail

OrgIDFilter

`public` **`OrgIDFilter`**`(int orgID)`

Constructs the filter based on a particular organization ID.

Parameters:

`orgID` – the Ooganization ID value for matching `RecordingRequest` instances.

Method Detail

getFilterValue

```
public int getFilterValue()
```

Reports the value of the organization ID used to create this filter.

Returns:

The organization ID used to filter.

---

accept

```
public boolean accept(RecordingRequest entry)
```

Tests if the given `RecordingRequest` passes the filter.

Specified by:

`accept` in class `RecordingListFilter`

Parameters:

`entry` – An individual `RecordingRequest` to be evaluated against the filtering algorithm.

Returns:

`true` if `RecordingRequest` is of the type indicated by the filter value; `false` otherwise.

D.2.4 RecordingList interface

```
org.ocap.shared.dvr.navigation
     Interface RecordingList
```

---

public interface **RecordingList**

RecordingList represents a list of recordings.

Method Summary	
boolean	**contains**(RecordingRequest entry) Tests if the indicated RecordingRequest object is contained in the list.
RecordingListIterator	**createRecordingListIterator**() Generates an iterator on the RecordingRequest elements in this list.
RecordingList	**filterRecordingList**(RecordingListFilter filter) Creates a new RecordingList object that is a subset of this list, based on the conditions specified by a RecordingListFilter object.
RecordingRequest	**getRecordingRequest**(int index) Reports the RecordingRequest at the specified index position.
int	**indexOf**(RecordingRequest entry) Reports the position of the first occurrence of the indicated RecordingRequest object in the list.
int	**size**() Reports the number of RecordingRequest objects in the list.
RecordingList	**sortRecordingList**(RecordingListComparator sortCriteria) Creates a new RecordingList that contains all the elements of this list sorted according to the criteria specified by a RecordingListComparator.

Method Detail

filterRecordingList

```
public RecordingList filterRecordingList(RecordingListFilter filter)
```

Creates a new RecordingList object that is a subset of this list, based on the conditions specified by a RecordingListFilter object. This method may be used to generate increasingly specialized lists of RecordingRequest objects based on multiple filtering criteria. If the filter is null, the resulting RecordingList will be a duplicate of this list.

Note that the accept method of the given RecordingListFilter will be invoked for each RecordingRequest to be filtered using the same application thread that invokes this method.

Parameters:

filter – A filter constraining the requested recording list, or null.

Returns:

A RecordingList object created based on the specified filtering rules.

---

createRecordingListIterator

```
public RecordingListIterator createRecordingListIterator()
```

Generates an iterator on the `RecordingRequest` elements in this list.

Returns:

A `RecordingListIterator` on the `RecordingRequest`s in this list.

---

contains

```
public boolean contains(RecordingRequest entry)
```

Tests if the indicated `RecordingRequest` object is contained in the list.

Parameters:

`entry` – The `RecordingRequest` object for which to search.

Returns:

`true` if the specified `RecordingRequest` is member of the list; `false` otherwise.

---

indexOf

```
public int indexOf(RecordingRequest entry)
```

Reports the position of the first occurrence of the indicated `RecordingRequest` object in the list.

Parameters:

`entry` – The `RecordingRequest` object for which to search.

Returns:

The index of the first occurrence of the `entry`, or `-1` if `entry` is not contained in the list.

---

size

```
public int size()
```

Reports the number of `RecordingRequest` objects in the list.

Returns:

The number of `RecordingRequest` objects in the list.

---

getRecordingRequest

```
public RecordingRequest getRecordingRequest(int index)
```

Reports the `RecordingRequest` at the specified index position.

Parameters:

`index` – A position in the `RecordingList`.

Returns:

The `RecordingRequest` at the specified index.

Throws:

`java.lang.IndexOutOfBoundsException` – If `index` < 0 or `index` > `size()-1`.

---

sortRecordingList

```
public RecordingList sortRecordingList(RecordingListComparator sortCriteria)
```

Creates a new `RecordingList` that contains all the elements of this list sorted according to the criteria specified by a `RecordingListComparator`.

Parameters:

`sortCriteria` – the sort criteria to be applied to sort the entries in the recording list.

Returns:

A sorted copy of the recording list.

D.2.5 RecordingListComparator interface

```
org.ocap.shared.dvr.navigation
     Interface RecordingListComparator
```

---

public interface **RecordingListComparator**

This interface represents a sorting criteria to be applied while sorting a RecordingList.

Method Summary	
`int`	**`compare`**`(RecordingRequest first, RecordingRequest second)` Compares two entries to check whether the first entry should be placed ahead of the second entry in the iterator list.

Method Detail

compare

```
public int compare(RecordingRequest first,
                   RecordingRequest second)
```

Compares two entries to check whether the first entry should be placed ahead of the second entry in the iterator list.

Parameters:

`first` – the first entry to compare

`second` – the second entry to compare

Returns:

positive integer if the first argument should be placed ahead of the second argument; negative integer if the second argument should be placed ahead of the first entry; zero if the current order should be retained.

D.2.6 RecordingListFilter class

```
org.ocap.shared.dvr.navigation
     Class RecordingListFilter

java.lang.Object
  +--org.ocap.shared.dvr.navigation.RecordingListFilter
```

Direct Known Subclasses:

AppIDFilter, OrgIDFilter, RecordingStateFilter

---

public abstract class **RecordingListFilter**

extends java.lang.Object

Base class for all RecordingListFilters. Subclasses of RecordingListFilter may be used to create filters to specify restrictions.

Constructor Summary	
protected	**RecordingListFilter**() Constructs the filter.

Method Summary	
abstract boolean	**accept**(RecordingRequest entry) Tests if a particular entry passes this filter.
void	**setCascadingFilter**(RecordingListFilter filter) Provides a means to cascade filters.

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

Constructor Detail

RecordingListFilter

```
protected RecordingListFilter()
```

Constructs the filter.

Method Detail

accept

```
public abstract boolean accept(RecordingRequest entry)
```

Tests if a particular entry passes this filter. Subtypes of `RecordingListFilter` override this method to provide the logic for a filtering operation on individual `RecordingRequest` objects.

Parameters:

`entry` – A `RecordingRequest` to be evaluated against the filtering algorithm.

Returns:

`true` if `entry` satisfies the filtering algorithm; `false` otherwise.

---

setCascadingFilter

`public void` **`setCascadingFilter`**`(RecordingListFilter filter)`

Provides a means to cascade filters. The accept method of this filter is called only for entries matching the specified filter. Multiple calls to this method will replace the previously set filter.

Parameters:

`filter` – the filter that will be applied before selecting the entries for which the accept() method is called. If the current filter is in the cascade chain of the filter passed in as the argument, this method does nothing.

D.2.7 RecordingListIterator interface

```
org.ocap.shared.dvr.navigation
     Interface RecordingListIterator
```

---

public interface **RecordingListIterator**

This iterator could be used to traverse entries in a RecordingList.

Method Summary	
RecordingRequest	**getEntry**(int index) Gets the RecordingRequest object at the specified position.
int	**getPosition**() Gets the current position of the RecordingListIterator.
int	**getPosition**(RecordingRequest entry) Gets the position of a specified recording request in the list.
RecordingList	**getRecordingList**() Gets the recording list corresponding to this RecordingListIterator.
boolean	**hasNext**() Tests if there is a RecordingRequest in the next position in the list.
boolean	**hasPrevious**() Tests if there is a RecordingRequest in the previous position in the list.
RecordingRequest[]	**nextEntries**(int n) Gets the next 'n' RecordingRequest objects in the list.
RecordingRequest	**nextEntry**() Gets the next RecordingRequest object in the list.
RecordingRequest[]	**previousEntries**(int n) Gets the previous 'n' RecordingRequest objects in the list.
RecordingRequest	**previousEntry**() Gets the previous RecordingRequest object in the list.
void	**setPosition**(int index) Sets the current position of the RecordingListIterator.
void	**toBeginning**() Resets the iterator to the beginning of the list, such that hasPrevious() returns false and nextEntry() returns the first RecordingRequest in the list (if the list is not empty).
void	**toEnd**() Sets the iterator to the end of the list, such that hasNext() returns false and previousEntry() returns the last RecordingRequest in the list (if the list is not empty).

Method Detail

toBeginning

public void **toBeginning**()

Resets the iterator to the beginning of the list, such that hasPrevious() returns false and nextEntry() returns the first RecordingRequest in the list (if the list is not empty).

---

toEnd

public void **toEnd**()

Sets the iterator to the end of the list, such that hasNext() returns false and previousEntry() returns the last RecordingRequest in the list (if the list is not empty).

---

nextEntry

`public RecordingRequest` **`nextEntry`**`()`

Gets the next `RecordingRequest` object in the list. This method may be called repeatedly to iterate through the list.

Returns:

The `RecordingRequest` object at the next position in the list.

Throws:

`java.util.NoSuchElementException` – If the iteration has no next `RecordingRequest`.

---

previousEntry

`public RecordingRequest` **`previousEntry`**`()`

Gets the previous `RecordingRequest` object in the list. This method may be called repeatedly to iterate through the list in reverse order.

Returns:

The `RecordingRequest` object at the previous position in the list.

Throws:

`java.util.NoSuchElementException` – If the iteration has no previous `RecordingRequest`.

---

hasNext

`public boolean` **`hasNext`**`()`

Tests if there is a `RecordingRequest` in the next position in the list.

Returns:

`true` if there is a `RecordingRequest` in the next position in the list; `false` otherwise.

---

hasPrevious

`public boolean` **`hasPrevious`**`()`

Tests if there is a `RecordingRequest` in the previous position in the list.

Returns:

`true` if there is a `RecordingRequest` in the previous position in the list; `false` otherwise.

---

nextEntries

`public RecordingRequest[]` **`nextEntries`**`(int n)`

Gets the next 'n' `RecordingRequest` objects in the list. This method also advances the current position within the list. If the requested number of entries is not available, the remaining elements are returned. If the current position is at the end of the iterator, this method returns an array with length zero.

Parameters:

`n` – the number of next entries requested.

Returns:

an array containing the next 'n' `RecordingRequest` object from the current position in the list.

---

previousEntries

```
public RecordingRequest[] previousEntries(int n)
```

Gets the previous 'n' `RecordingRequest` objects in the list. This method also changes the current position within the list. If the requested number of entries is not available, the remaining elements are returned. If the current position is at the beginning of the iterator, this method returns an array with length zero.

Parameters:

`n` – the number of previous entries requested.

Returns:

an array containing the previous 'n' `RecordingRequest` object from the current position in the list.

---

getEntry

```
public RecordingRequest getEntry(int index)
```

Gets the `RecordingRequest` object at the specified position. This method does not advance the current position within the list.

Parameters:

`index` – the position of the RecordingRequest to be retrieved.

Returns:

the `RecordingRequest` at the specified position.

Throws:

`java.lang.IndexOutOfBoundsException` – if the index is greater than the size of the list.

---

getPosition

```
public int getPosition(RecordingRequest entry)
```

Gets the position of a specified recording request in the list.

Parameters:

`entry` – The recording request for which the position is sought.

Returns:

The position of the specified recording; –1 if the entry is not found.

---

getPosition

```
public int getPosition()
```

Gets the current position of the RecordingListIterator. This would be the position from where the next RecordingRequest will be retrieved when an application calls the nextEntry.

Returns:

the current position of the RecordingListIterator.

---

setPosition

```
public void setPosition(int index)
                 throws java.lang.IndexOutOfBoundsException
```

Sets the current position of the RecordingListIterator. This would be the position from where the next RecordingRequest will be retrieved when an application calls the nextEntry.

Parameters:

`index` – the current position of the RecordingListIterator would be set to this value.

Throws:

`java.lang.IndexOutOfBoundsException` – if the index is greater than the size of the list.

---

getRecordingList

```
public RecordingList getRecordingList()
```

Gets the recording list corresponding to this RecordingListIterator.

Returns:

the RecordingList corresponding to this iterator.

D.3 Shared media package

```
org.ocap.shared.media
     Package
```


D.3.1 TimeShiftControl interface
D.3.2 BeginningOfContentEvent class
D.3.3 EndOfContentEvent class
D.3.4 EnteringLiveModeEvent class
D.3.5 LeavingLiveModeEvent class
D.3.6 MediaTimeFactoryControl interface
D.3.7 TimeLine interface
D.3.8 TimeLineControl interface
D.3.9 TimeLineInvalidException class
D.3.10 TimeOutOfRangeException class

D.3.1 TimeShiftControl interface

```
org.ocap.shared.media
     Interface TimeShiftControl
```

All Superinterfaces:

javax.media.Control

---

public interface **TimeShiftControl**

extends javax.media.Control

This interface represents a trick-mode control that can be used for retrieving more information corresponding to the playback of the time-shift buffer. This control will only be available is the service being presented on the service context is a broadcast service and if there is a time-shift buffer associated with the service context.

Method Summary	
`javax.media.Time`	**`getBeginningOfBuffer`**`()` Gets the media time corresponding to the current beginning of the time-shift buffer.
`javax.media.Time`	**`getDuration`**`()` Gets the duration of content currently in the time-shift buffer.
`javax.media.Time`	**`getEndOfBuffer`**`()` Gets the media time corresponding to the end of the time-shift buffer.
`javax.media.Time`	**`getMaxDuration`**`()` Gets the estimated value for the maximum duration of content that could be buffered using this time-shift buffer.

Methods inherited from interface javax.media.Control
`getControlComponent`

Method Detail

getBeginningOfBuffer

```
public javax.media.Time getBeginningOfBuffer()
```

Gets the media time corresponding to the current beginning of the time-shift buffer. This could be the media time corresponding to start of the buffer, before the buffer wrap around or the media time corresponding to the beginning of the valid buffer area after the wrap around.

Returns:

media time corresponding to the beginning of the time-shift buffer.

---

getEndOfBuffer

```
public javax.media.Time getEndOfBuffer()
```

Gets the media time corresponding to the end of the time-shift buffer. This could be the current system time if the time-shift recording is still ongoing or the media time corresponding to the end point for the valid area of the time-shift buffer.

Returns:

media time corresponding to the end of the time-shift buffer.

---

getDuration

```
public javax.media.Time getDuration()
```

Gets the duration of content currently in the time-shift buffer. The value returned is the content's duration when played at a rate of 1.0.

Returns:

A Time object representing the duration.

---

getMaxDuration

```
public javax.media.Time getMaxDuration()
```

Gets the estimated value for the maximum duration of content that could be buffered using this time-shift buffer. The value returned is the content's duration when played at a rate of 1.0.

Returns:

A Time object representing the maximum value for duration.

D.3.2 BeginningOfContentEvent class

```
org.ocap.shared.media
     Class BeginningOfContentEvent

java.lang.Object
  +--java.util.EventObject
       +--javax.media.ControllerEvent
            +--javax.media.RateChangeEvent
                 +--org.ocap.shared.media.BeginningOfContentEvent
```

All Implemented Interfaces:

javax.media.MediaEvent, java.io.Serializable

---

public class **BeginningOfContentEvent**

extends javax.media.RateChangeEvent

BeginningOfContentEvent is a RateChangeEvent that is posted when the rate change is due to a rewind hitting the beginning of the media, or due to the time-shift buffer reaching maximum depth.

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.EventObject

source

Constructor Summary

BeginningOfContentEvent(javax.media.Controller from, float newRate) Creates a BeginningOfContentEvent.

Methods inherited from class javax.media.RateChangeEvent

getRate, toString

Methods inherited from class javax.media.ControllerEvent

getSource, getSourceController

Methods inherited from class java.lang.Object

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

Constructor Detail

BeginningOfContentEvent

```
public BeginningOfContentEvent(javax.media.Controller from,
                               float newRate)
```

Creates a BeginningOfContentEvent.

Parameters:

`from` – the controller that is generating the event.

D.3.3 EndOfContentEvent class

```
org.ocap.shared.media
     Class EndOfContentEvent

java.lang.Object
  +--java.util.EventObject
      +--javax.media.ControllerEvent
          +--javax.media.RateChangeEvent
              +--org.ocap.shared.media.EndOfContentEvent
```

All Implemented Interfaces:

javax.media.MediaEvent, java.io.Serializable

---

public class **EndOfContentEvent**

extends javax.media.RateChangeEvent

EndOfContentEvent is a RateChangeEvent that is posted when the rate change is due to a forward playback hitting the end of the stored context, or a forward playback catching up with the live recording point.

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.EventObject

`source`

Constructor Summary

`EndOfContentEvent``(javax.media.Controller from, float newRate)` Creates an EndOfContentEvent.

Methods inherited from class javax.media.RateChangeEvent

`getRate, toString`

Methods inherited from class javax.media.ControllerEvent
getSource, getSourceController

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

Constructor Detail

EndOfContentEvent

```
public EndOfContentEvent(javax.media.Controller from,
                         float newRate)
```

Creates an EndOfContentEvent.

Parameters:

`from` – the controller that is generating the event.

D.3.4 EnteringLiveModeEvent class

```
org.ocap.shared.media
     Class EnteringLiveModeEvent

java.lang.Object
  +--java.util.EventObject
      +--javax.media.ControllerEvent
          +--org.ocap.shared.media.EnteringLiveModeEvent
```

All Implemented Interfaces:

javax.media.MediaEvent, java.io.Serializable

---

public class **EnteringLiveModeEvent**

extends javax.media.ControllerEvent

EnteringLiveModeEvent is a ControllerEvent that is posted when the controller has started playing back a live broadcast stream. This event is sent to a registered ControllerListener in addition to any RateChangeEvent or MediaTimeSetEvent.

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.EventObject
source

Constructor Summary
EnteringLiveModeEvent(javax.media.Controller from) Creates an EnteringLiveModeEvent.

Methods inherited from class javax.media.ControllerEvent
getSource, getSourceController, toString

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

Constructor Detail

EnteringLiveModeEvent

public **EnteringLiveModeEvent**(javax.media.Controller from)

Creates an EnteringLiveModeEvent.

Parameters:

from – the controller that is generating the event.

D.3.5 LeavingLiveModeEvent class

```
org.ocap.shared.media
     Class LeavingLiveModeEvent

java.lang.Object
  +--java.util.EventObject
       +--javax.media.ControllerEvent
            +--org.ocap.shared.media.LeavingLiveModeEvent
```

All Implemented Interfaces:

javax.media.MediaEvent, java.io.Serializable

---

public class **LeavingLiveModeEvent**

extends javax.media.ControllerEvent

LeavingLiveModeEvent is a ControllerEvent that is posted when the controller is not playing back a live broadcast stream anymore. This event is sent to a registered ControllerListener in addition to any RateChangeEvent or MediaTimeSetEvent.

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.EventObject
source

Constructor Summary
LeavingLiveModeEvent(javax.media.Controller from) Creates a LeavingLiveModeEvent.

Methods inherited from class javax.media.ControllerEvent
getSource, getSourceController, toString

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

Constructor Detail

LeavingLiveModeEvent

public **LeavingLiveModeEvent**(javax.media.Controller from)

Creates a LeavingLiveModeEvent.

Parameters:

from – the controller that is generating the event.

D.3.6 MediaTimeFactoryControl interface

org.ocap.shared.media
Interface MediaTimeFactoryControl

All Superinterfaces:

javax.media.Control

---

public interface **MediaTimeFactoryControl**

extends javax.media.Control

Provides the ability to obtain media times with various special characteristics when applied to the content being played by this JMF player. The behaviour of these media times is implementation dependent if used with any other JMF player.

Method Summary	
javax.media.Time	**getRelativeTime**(long offset) Obtains a media time relative to the current location
javax.media.Time	**setTimeApproximations**(javax.media.Time original, boolean beforeOrAfter) Enables applications to precisely control the position where playback starts following a call to Player.setMediaTime.

Methods inherited from interface javax.media.Control
getControlComponent

Method Detail

getRelativeTime

```
public javax.media.Time getRelativeTime(long offset)
```

Obtains a media time relative to the current location

Parameters:

offset – the offset relative to the current location measured in nanoseconds

Returns:

a media time

---

setTimeApproximations

```
public javax.media.Time setTimeApproximations(javax.media.Time original,
                                              boolean beforeOrAfter)
```

Enables applications to precisely control the position where playback starts following a call to Player.setMediaTime. This method takes an original media time as input and returns a new media time which encapsulates the original media time and an indication of how that original media time is to be interpreted when used in a call to Player.setMediaTime.

Parameters:

original – the original media time

beforeOrAfter – if true, the media time where playback starts will be on or before the original one (i.e., content at the original media time is guaranteed to be presented in playback). If false, the media time where playback starts will be after the original one (i.e., neither content at the original media time nor any content before that original time will be presented in playback).

Returns:

a new media time

D.3.7 TimeLine interface

```
org.ocap.shared.media
     Interface TimeLine
```

---

public interface **TimeLine**

Represents a transmitted time line. Transmitted time lines start at one media time within a piece of content and finish at a later media time in that content. Transmitted time lines are valid at all media times between these points. They are either incremented linearly or are paused. The value of a transmitted time line does not have any discontinuities.

Method Summary	
`javax.media.Time`	**`getFirstMediaTime`**`()` Returns the first media time at which this time line is valid.
`long`	**`getFirstTime`**`()` Returns the first valid time in this time line.
`javax.media.Time`	**`getLastMediaTime`**`()` Returns the last time at which this time line is valid.
`long`	**`getLastTime`**`()` Returns the last valid time in this time line.
`javax.media.Time`	**`getMediaTime`**`(long time)` Translates a time in this time line into the corresponding media time.
`long`	**`getTime`**`(javax.media.Time mediatime)` Translates a media time into the corresponding time in this timeline

Method Detail

getFirstMediaTime

```
public javax.media.Time getFirstMediaTime()
                                    throws TimeLineInvalidException
```

Returns the first media time at which this time line is valid. For a scheduled recording, this is the first point within the piece of content where the time line is valid. For a timeshift recording, if the time line starts within the time shift buffer then the media time where it starts will be returned. If the time line starts before the start of the time shift buffer, the media time of the start of the time shift buffer will be returned. Note that if the time shift buffer is full and time shift recording is in progress, the start of the buffer will be moving as newly written data overwrites the former start of the buffer.

Returns:

a media time

Throws:

`TimeLineInvalidException` – if the time line is no longer valid in this piece of content. E.g., the piece of content is a time shift recording and the end of the time line is no longer within the buffer

---

getLastMediaTime

```
public javax.media.Time getLastMediaTime()
                                    throws TimeLineInvalidException
```

Returns the last time at which this time line is valid. For a scheduled recording, this is the last point within the piece of content where the time line is valid. For a timeshift recording, if the time line ends within the time shift buffer then the media time where it starts will be returned. If the time line ends after the end of the time shift buffer, the media time of the end of the time shift buffer will be returned. Note that if the time shift buffer is full and time shift recording is in progress, the end of the buffer will be moving as newly written data overwrites the former start of the buffer.

Returns:

a media time

Throws:

`TimeLineInvalidException` – if the time line is no longer valid in this piece of content. E.g., the piece of content is a time shift recording and the end of the time line is no longer within the buffer

---

getFirstTime

```
public long getFirstTime()
                  throws TimeLineInvalidException
```

Returns the first valid time in this time line. For a scheduled recording, this is the first point within the piece of content where the time line is valid. For a timeshift recording, if the time line starts within the time shift buffer then the time where it starts will be returned. If the time line starts before the start of the time shift buffer, the time of the start of the time shift buffer will be returned. Note that if the time shift buffer is full and time shift recording is in progress, the start of the buffer will be moving as newly written data overwrites the former start of the buffer.

Returns:

a time in this time line

Throws:

`TimeLineInvalidException` – if the time line is no longer valid in this piece of content. E.g., the piece of content is a time shift recording and the end of the time line is no longer within the buffer

---

getLastTime

```
public long getLastTime()
                  throws TimeLineInvalidException
```

Returns the last valid time in this time line. For a scheduled recording, this is the last point within the piece of content where the time line is valid. For a timeshift recording, if the time line ends within the time shift buffer then the media time where it end will be returned. If the media time ends after the end of the time shift buffer, the media time of the end of the time shift buffer will be returned. Note that if the time shift buffer is full and time shift recording is in progress, the end of the buffer will be moving as newly written data overwrites the former start of the buffer.

Returns:

a time in this time line

Throws:

`TimeLineInvalidException` – if the time line is no longer valid in this piece of content. E.g., the piece of content is a time shift recording and the end of the time line is no longer within the buffer

---

getMediaTime

```
public javax.media.Time getMediaTime(long time)
                              throws TimeLineInvalidException,
                                     TimeOutOfRangeException
```

Translates a time in this time line into the corresponding media time. If the time is one where the time line pauses, the returned media time shall be the highest media time corresponding to the time specified.

Parameters:

`time` – a time in this time line

Returns:

the corresponding media time

Throws:

`TimeLineInvalidException` – if the time line is no longer valid in this piece of content. E.g., the piece of content is a time shift recording and the end of the time line is no longer within the buffer

`TimeOutOfRangeException` – if the time specified is not within this timeline

---

getTime

```
public long getTime(javax.media.Time mediatime)
             throws TimeLineInvalidException,
                    TimeOutOfRangeException
```

Translates a media time into the corresponding time in this timeline

Parameters:

`mediatime` – a media time

Returns:

the corresponding time in this timeline

Throws:

`TimeLineInvalidException` – if the time line is no longer valid in this piece of content. E.g., the piece of content is a time shift recording and the end of the time line is no longer within the buffer

`TimeOutOfRangeException` – if the media time specified is not within this timeline

D.3.8 TimeLineControl interface

```
org.ocap.shared.media
     Interface TimeLineControl
```

All Superinterfaces:

javax.media.Control

---

public interface **TimeLineControl**

extends javax.media.Control

Provides access to the transmitted timelines in a piece of content

Method Summary	
`TimeLine[]`	**`getTimeLines`**`()` Returns all the transmitted timelines found in a piece of content.

Methods inherited from interface javax.media.Control
`getControlComponent`

Method Detail

getTimeLines

```
public TimeLine[] getTimeLines()
```

Returns all the transmitted timelines found in a piece of content. If no transmitted timelines are present then an array of length 0 is returned.

Returns:

an array of timelines

D.3.9 TimeLineInvalidException class

```
org.ocap.shared.media
     Class TimeLineInvalidException

java.lang.Object
  +--java.lang.Throwable
       +--java.lang.Exception
           +--org.ocap.shared.media.TimeLineInvalidException
```

All Implemented Interfaces:

java.io.Serializable

---

public class **TimeLineInvalidException**

extends java.lang.Exception

This exception is returned when a time line is no longer valid in the piece of content for which it was obtained. For example, the piece of content is a time shift recording and the end of the time line is no longer within the buffer.

See Also:

Serialized Form

Constructor Summary
TimeLineInvalidException() Constructs a TimeLineInvalidException with no detail message

Methods inherited from class java.lang.Throwable
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

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

Constructor Detail

TimeLineInvalidException

```
public TimeLineInvalidException()
```

Constructs a TimeLineInvalidException with no detail message

D.3.10 TimeOutOfRangeException class

```
org.ocap.shared.media
     Class TimeOutOfRangeException

java.lang.Object
  +--java.lang.Throwable
       +--java.lang.Exception
            +--org.ocap.shared.media.TimeOutOfRangeException
```

All Implemented Interfaces:

java.io.Serializable

---

public class **TimeOutOfRangeException**

extends java.lang.Exception

This exception is returned when a time or media time is outside the valid range for a particular time line.

See Also:

Serialized Form

Constructor Summary
`TimeOutOfRangeException``()` Constructs a TimeOutOfRangeException with no detail message

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString`

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

Constructor Detail

TimeOutOfRangeException

`public` **`TimeOutOfRangeException`**`()`

Constructs a TimeOutOfRangeException with no detail message

Bibliography

[b-DAVIC 1.4.1p9] DAVIC 1.4.1p9, DAVIC 1.4.1 Specification Part 9 – *Information Representation*.

[b-ETSI TS 102 816] ETSI TS 102 816, *Digital Video Broadcasting (DVB); PVR/PDR Extension to the Multimedia Home Platform.*

NOTE – At the time of publication of this Recommendation, the above reference is only available as DVB Bluebook A088.rev1. It will become available from ETSI in due course.

[b-OC-SP-OCAP-DVR-I02-050524] OC-SP-OCAP-DVR-I02-050524, *OpenCable Application Platform Specification; OCAP Digital Video Recorder (DVR)*.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Cable networks and transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2008