International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.215

(12/2007)

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

Interactive systems for digital television distribution

Client digital program insertion API

Recommendation ITU-T J.215

ITU-T

Recommendation ITU-T J.215

Client digital program insertion API

Summary

The digital program insertion (DPI) Recommendations given in ITU-T J.181 and ITU-T J.280 have enabled cable operators to provide local ad insertion on digital content at a head-end level, using equipment available from multiple vendors. Recommendation ITU-T J.215 enables similar functionality for DPI including addressable commercials, or other content, to be inserted at a device level in the consumer premises equipment.

Source

Recommendation ITU-T J.215 was approved on 14 December 2007 by ITU-T Study Group 9 (2005-2008) under the ITU-T Recommendation A.8 procedures.

CONTENTS

Recommendation ITU-T J.215

Client digital program insertion API

1 Scope

This Recommendation defines requirements for digital program insertion, henceforth referred to as DPI.

This recommendation enables digital program insertion in host devices ("client devices") within customers' homes, enabling addressable commercials to be inserted and addressed at a household level (or even at an individual television set level).

2 References

2.1 Normative 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]	Recommendation ITU-T J.202 (2005), *Harmonization of procedural content formats for interactive TV applications*.
[ITU-T J.280]	Recommendation ITU-T J.280 (2005), *Digital Program Insertion: Splicing application program interface*.
[DVB-SAD]	ETSI TS 102 823 v1.1.1 (2005), *Digital Video Broadcasting (DVB); Specification for the carriage of synchronized auxiliary data in DVB transport streams,* ‹http://pda.etsi.org/pda/queryform.asp›.
[ANSI/SCTE 138]	ANSI/SCTE 138 (2009), *Stream Conditioning for Switching of Addressable Content in Digital Television Receivers*.

2.2 Informative references

[ITU-T J.181]	Recommendation ITU-T J.181 (2004), *Digital program insertion cueing message for cable television systems*.
[ITU-T J.201]	Recommendation ITU-T J.201 (2004), *Harmonization of declarative content format for interactive television applications*.

3 Definitions

This Recommendation defines the following terms:

3.1 application: An application is a functional implementation realized as software running in one or spread over several interplaying hardware entities.

3.2 application program interface (API): An application program interface is the software interface to system services or software libraries. An API can consist of classes, function calls, subroutine calls, descriptive tags, etc.

3.3 broadcast: A broadcast is a service that is delivered to all customers. Each customer may select a particular broadcast channel out of many.

3.4 broadcast application: A broadcast application is an application running on the set-top converter that is loaded through in-band information, inserted either at the head-end or by a content provider farther upstream.

3.5 client device: The CPE that is connected to the cable network in the consumer's home, receives the television signals at the client's premises, and presents it for display on a display device.

3.6 digital program insertion (DPI): Insertion of alternative content into digitally encoded content in response to messaging in the stream.

3.7 packet identifier (PID): MPEG-2 assigns a PID to each data packet. Packets with the same PID belong to the same logical channel.

3.8 program map table (PMT): This is a MPEG-2 entity that contains all of the PIDs that make up a program.

3.9 switch engine: This term refers to the functionality that executes switching in the host device. This switching can either be seamless, or non-seamless. Because this functionality needs to be implemented in a real-time way, with predictable timing behaviour, the switching engine is implemented as part of an OpenCable Applications Platform (OCAP) implementation ("below the line"). It exposes an API to OCAP applications (targeting engines) to select which commercials are switched to/from. A switching engine can be implemented in hardware, software, or a combination of both.

3.10 unicast advertising: Advertising content directed at a single target or small group of targets.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations:

API	Application Program Interface
DPI	Digital Program Insertion
OCAP	OpenCable Applications Platform
PMT	Program Map Table

5 Conventions

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

SHALL	This word, or the adjective REQUIRED, means that the item is an absolute requirement of this Recommedation.
SHALL NOT	This phrase means that the item is an absolute prohibition of this Recommendation.
SHOULD	This word, or the adjective RECOMMENDED, means that valid reasons may exist in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
SHOULD NOT	This phrase means that there may exist valid reasons in particular circumstances when the listed behaviour is acceptable or even useful, but the

full implications should be understood and the case carefully weighed before implementing any behaviour described with this label.

MAY	This word, or the adjective OPTIONAL, means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

6 DPI overview

The digital program insertion (DPI) standards within [ITU-T J.181] and [ITU-T J.280] have enabled cable operators to provide local ad insertion on digital content at a head-end level, using equipment available from multiple vendors. This Recommendation enables similar functionality for DPI including addressable commercials, or other content, to be inserted at a device level.

To allow for programmability by a cable operator as well as meet time sensitive insertion requirements, the DPI functionality is separated into two logical components: a 'switch engine' implemented as part of the J.201 platform that performs real-time functions of switching between programs and program elements, and one or more 'targeting engine' applications that control the behaviour of the switch engine.

A switch engine performs switch operations to present inserted content. In the broadcast scenario, which is the only scenario defined herein, inserted content MAY be within the same MPEG-2 transport stream as the original program or broadcast within a different MPEG-2 transport stream. Other scenarios, such as those where the original or inserted content are available on a storage device, may be defined elsewhere.

A switch engine implements two distinct types of switches; a Level 0 (L0) switch, which appears to the viewer as a channel change, and a Level 1 (L1) switch, which is totally invisible to the viewer. During an L0 switch, a switch engine may be required to tune to a different channel to access an insertion stream. An L1 switch never requires tuning; in this case, all of the insertion streams are within the same transport stream as the original program content.

Requirements on content to enable L0 and L1 switches are defined by [ANSI/SCTE 138]. The switch engine behaviour is defined in clause 7.2. Detailed requirements on OCAP to perform L0 and L1 switches are defined in Annex A.

An illustration of the logical components of the OCAP DPI capability is provided in Annex A.

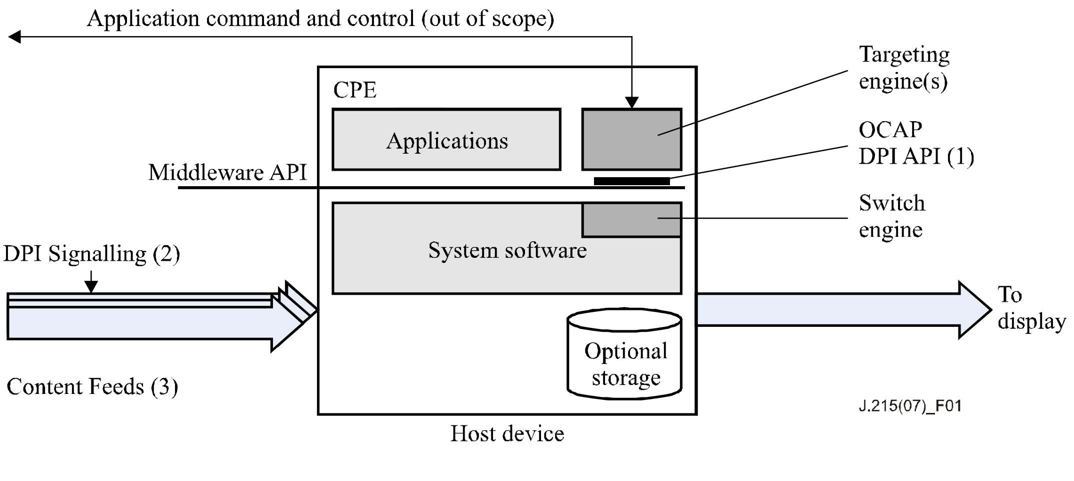

Figure 1 – OCAP DPI overview

Indicated as (1) in Figure 1, a targeting engine API is defined Annex A to enable applications to provide the switch engine with instructions and to receive status information on previously submitted instructions.

Indicated as (2) in Figure 1, DPI signalling are messages that are processed by the switch engine. Signalling may be in the form of DPI triggers or timelines. See clause 7.1 below for detailed requirements on processing DPI signalling.

Where instructed by a targeting engine, the switch engine performs a program insertion operation on receipt of a DPI trigger, or based on a timeline. The DPI API provides the mechanism by which a targeting engine application configures the switch engine to perform a switch operation. A switch operation occurs when the switch engine is properly configured by a targeting application through the DPI API, and the proper signalling is subsequently detected.

Indicated as (3) in Figure 1, content feeds are MPEG-2 transport streams that carry program feeds and may contain other insertion feeds, such as advertising feeds. The program feeds (as well as the separate insertion feeds) may contain DPI signalling that is used by the switch engine to insert content, such as addressable commercials during addressable breaks. The program feeds and insertion feeds may be subject to certain encoding (and other) restrictions that make it easier for the switching engine to switch to and from insertion content, see [ANSI/SCTE 138].

The command and control element shown in Figure 1 represents a hypothetical private communication channel between a targeting engine and an associated network component. Advanced advertising systems are expected to incorporate logical components within the cable network that manage both the insertion of DPI signalling and the actions of a targeting engine.

No security model has been defined to protect signalling from detection by non-authorized users. Because of the extremely tight synchronization between signalling and video content needed to perform seamless L1 switches, signalling must be in-band, in the same program stream as the original content feed. Strategies for obfuscating signals may be developed, but none are described herein.

7 Requirements for digital program insertion (DPI)

7.1 DPI signalling

This clause describes signaling used to enable a switch engine to perform its operations.

Two signalling protocols are used to support DPI: DPI triggers and DPI timelines. DPI trigger messages embedded within a content segment indicate opportunities to perform a switch operation. DPI timelines are messages embedded within a content segment to enable the implementation to maintain a metadata timeline for a segment. Switch operations MAY be based on timelines.

All DPI signalling is expected to be carried on a single data PID within the original content stream. This data stream SHALL be identified by DPI registration descriptor in a program's PMT.

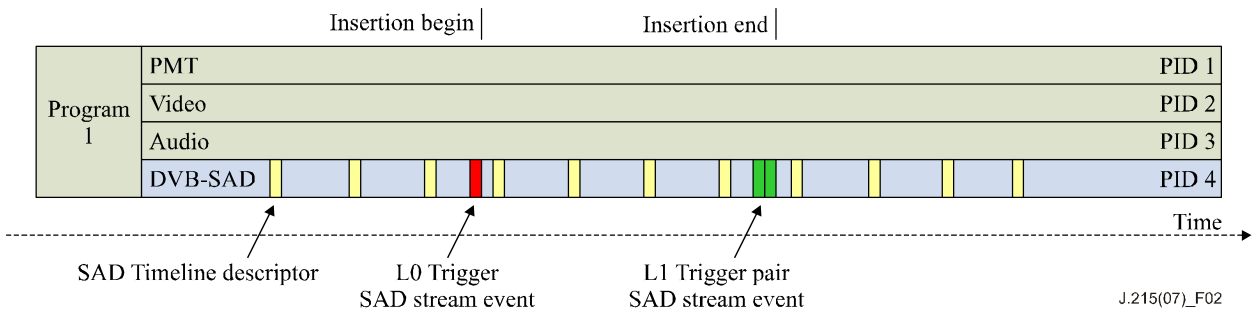

Figure 2 – DPI signalling stream

Figure 2 illustrates a hypothetical MPEG-2 program stream containing DPI signalling. The DVB-SAD PID contains a series of timeline descriptors, and examples of L0 and L1 triggers. The details of these descriptors and their placement are provided below.

7.1.1 DPI Signalling descriptor

The `dpi_signaling_descriptor` is defined for use in the elementary stream information loop of the PMT. This descriptor indicates that the associated elementary stream contains a DPI signalling stream, containing DPI triggers and timeline messages as defined below. Only one elementary stream signalled by the PMT SHALL contain a `dpi_signaling_descriptor`. In the event that more than one PMT entry contains a `dpi_signaling_descriptor`, the implementation SHALL ignore all but the first instance listed in the PMT's elementary stream information loop.

A PMT entry with a `dpi_signaling_descriptor` MAY be associated with a stream_type of 0xC0 or 0x05.

The `dpi_signaling_descriptor` is defined in Table 1.

Table 1 – DPI signalling descriptor syntax

Syntax	Bits	Mnemonic
`dpi_signaling_descriptor() {`		
`    descriptor_tag`	8	uimsbf
`    descriptor_length`	8	uimsbf
`    For (i=0; i<n; i++) {`		
`        private_use`	8	bslbf
`    }`		
`}`		

descriptor_tag	This 8-bit integer with value 0xA2 identifies this descriptor.
descriptor_length	This 8-bit integer indicates the number of bytes following the descriptor length field.
private_use	This field MAY be used to carry private data to a receiver which interprets this signalling stream. Its use is not defined by this Recommendation.

7.1.2 DPI triggers

The format for DPI triggers SHALL be the DVB-SAD synchronized event descriptor, see section 5.2.5 of [DVB-SAD]. When present in a content or insertion feed, the implementation SHALL process every instance of a DVB-SAD synchronized event descriptor.

The use of DPI triggers is different for L0 and L1 switches. To signal an L0 switch, a single DPI trigger MAY be used to indicate the point in a stream at which the switch may occur. For L1 switches, a DPI trigger SHALL be present for each component that will be switched; typically, one for video and one for audio. For L1 switches, a set of DPI triggers SHALL be used to signal a single switch operation.

The following semantics are defined for the DVB-SAD synchronized event descriptor (section 5.2.5 of [DVB-SAD]):

synchronized_event_context: This field SHALL be encoded according to Table 2. If this field contains a value not defined below, the implementation SHALL ignore this descriptor.

Table 2 – synchronized_event_context

Value	Description
10	L1 video
11	L1 audio
12	L0

synchronized_event_id: This field identifies a switch operation. The implementation SHALL use this value to associate a switch instruction with this trigger, if such an instruction has been received from an application.

synchronized_event_data_byte: This field SHALL be encoded according to Table 3. If this field is not conformant to Table 3, the implementation SHALL ignore this descriptor.

Table 3 – synchronized_event_data_byte

Syntax	No. of bits	Identifier
set_size	8	uimsbf
position_in _set	8	uimsbf

set_size: This field indicates the number of triggers signalling a switch. For an L0 switch, this value is 1, for an L1 switch, this value is at least 1 and is typically 2; one for video, one for audio.

position_in_set: This field indicates the position of this trigger within a set of triggers, zero based.

DPI triggers for L0 switches SHALL be present in a stream at a point TBD.

DPI triggers for L1 switches SHALL be present in a stream at a point TBD. For more details on the structure of an L1 conditioned stream, see [DVB-SAD].

7.1.3 DPI timelines

A content segment MAY contain timeline messages that allow the implementation to maintain a timeline synchronized to the video and audio content. Timeline messages SHALL be in the DVB-SAD broadcast timeline descriptor format, see [DVB-SAD] clause 5.2.2 for a description of the broadcast timeline descriptor. When present within a content feed, the implementation SHALL process every instance of a DVB-SAD broadcast timeline descriptor to maintain an interpolated metadata timeline. [DVB-SAD] contains descriptions and references to definitions of timelines within several clauses. The reader should be familiar with [DVB-SAD] in its entirety.

L0 switch operations MAY be performed when a point on a timeline is reached. In order to enable time-based switch scenarios in which a viewer may tune away, then tune back into a service during a switch opportunity, streams MAY contain a series of DPI timeline messages throughout the original program.

Within a content feed that signals a timeline and supports L0 switches, a DPI timeline SHALL be continuous throughout the relevant segment. The associated insertion feeds SHALL signal a timeline that is proceeding at the same rate with the same values as the original feed.

Within a content feed that signals a timeline and supporting L1 switches, the DPI signalling stream SHALL contain a continuously advancing timeline.

The following semantics are defined for the DVB-SAD synchronized event descriptor:

Where broadcast_timeline_type equals 0:

tick_format: This field SHALL have the value 0x11. See Table 6 of [DVB-SAD].

7.2 Switch engine behaviour

Switches may be trigger-based or time-based. A switch instruction MAY identify a set of triggers, and or identify a segment of time during which a switch may occur. A switch is either from an original program feed to an insertion feed, from an insertion feed to another insertion feed, or from an insertion feed back to the original feed. The switch engine SHALL maintain context such that the original feed may be presented in error cases or as default behaviour when the final insertion within a group is completed.

On receipt of a DPI trigger, the implementation SHALL determine whether there is a switch instruction matching the trigger. To match a switch instruction with a trigger, the implementation SHALL determine whether there is a SwitchInstruction with the switchID attribute equal to the value in the synchronized_event_id field of the trigger.

Insertions are often made in groups, as is the case when several ad spots make up an ad break. A switch instruction MAY indicate that if a previous insertion within a group is not made, then subsequent insertions should not take place, as might be the case if a service is selected after the first spot in an ad break. An instruction indicates that the entire group must be inserted when the insertionIndivisible attribute is non-zero, and indicates its position within a group via the groupSize and positionInGroup attributes. Where a matched instruction's insertionIndivisible attribute is non-zero, and all previous instructions within its group have not caused a switch, the implementation SHALL not perform a switch. Requirements defined below for performing switches assume this behaviour and do not repeat these statements.

Where an instruction matches a trigger:

- Where the value in the synchronized_event_context field of the trigger indicates an L0 trigger, the implementation SHALL perform an L0 switch.
- Where the value in the synchronized_event_context field of the trigger indicates an L1 operation, the implementation SHALL perform an L1 switch.

For time-based switches, the implementation will process DPI timeline messages if present and maintain a metadata timeline for the current service. See clause 7.1.3 for details. When the point on the metadata timeline is reached that is equal to or greater than the value of the switchStartTime (if non-zero) of any switch instruction for the service, and the triggerSwitch flag of the switch instruction is zero, the implementation SHALL perform an L0 switch.

7.2.1 Requirements to perform an L0 switch

The implementation SHALL present the program specified by the switch instruction destination field or by the default rules defined for the end of an insertion. Note that an L0 DPI switch is not a service selection, that is, an L0 switch does not affect the lifecycle of any applications associated with the selected service of the original channel, while it may cause the loading and launching of applications associated with the newly presenting program.

7.2.2 Requirements to perform an L1 switch

L1 switches are signalled by a set of DPI triggers, one for each component that will be replaced, i.e., one for video, one for audio, etc. L1 triggers are identified as such by the synchronized_event_context field of the trigger, see clause 7.1.2.

When an L1 video trigger matching a switch instruction is received, the implementation SHALL discontinue processing the video component of the currently presenting program and begin processing the video component of the program indicated by the destination field of the instruction.

When an L1 audio trigger matching a switch instruction is received, the implementation SHALL discontinue processing the audio component of the currently presenting program and begin processing the first audio component indicated in the PMT of the program indicated by the destination field of the instruction, or the audio component indicated by the instructions audioChannel attribute if non-zero.

The latency between discontinuation of processing an original component and processing a destination component SHALL NOT be less than one millisecond and SHALL NOT exceed 30 milliseconds.

When a set of video and audio triggers has been processed, the implementation SHALL process all data streams associated with the destination program. This is for the purpose of launching applications that may be associated with the destination program. Note that the destination program may be the original program in the case where the switch is from an insertion feed back to the original feed.

7.2.3 Selecting a DPI enabled program

The following requirements for maintaining timelines and performing time-based switches when a service is selected are to accommodate scenarios in which a viewer tunes away from a program and tunes back during an insertion opportunity.

Where a metadata timeline has been established for a service, see clause 7.1.3, the implementation SHALL maintain the timeline if there is an instruction for the service with non-zero values for switchStartTime and switchEndTime and another service is selected. If a third service is selected, the implementation MAY discontinue maintaining the timeline for the original service. If the original service is not re-selected within an hour, the implementation MAY discontinue maintaining the timeline.

If a service is selected and its timeline is active, and the value of the timeline is greater than or equal to the switchStartTime and less than or equal to the switchWindowEndTime, the implementation SHALL perform an L0 switch; an exception to this requirement is where switchWindowEndTime is within four seconds of the switchEndTime: the implementation SHALL not perform an L0 switch within four seconds of the switchEndTime.

7.2.4 Switch engine events

If a trigger is encountered and there is no corresponding instruction, an UNARMED_TRIGGER event SHALL be propagated to any listening targeting engine. If an insertion feed is currently being presented, the switch engine SHALL revert to presenting the original feed.

Where a switch occurs, an INSERTION_START event SHALL be propagated to any listening targeting engine.

If a viewer selects a DPI enabled service at a point between triggers within a set of L1 triggers, the receiver will not receive all of the triggers within the set, and therefore SHALL NOT perform a switch. Specifically, if the position_in_set field of a trigger is greater than or equal to two and a trigger has not been received on the service with position_in_set equal to 1, all triggers within the set SHALL be ignored. If not already presenting the original program, the implementation SHALL switch back to the original program and a MISSED_TRIGGER event SHALL be propagated to any listening targeting engine.

When a switch instruction expires (see Annex A), an INSTRUCTION_EXPIRED event SHALL be propagated to any targeting engine, and the instruction SHALL be considered inactive.

7.3 Switch Engine API

This clause describes the functionality of an application programming interface to a switch engine. This description is provided in 'C' syntax, and can be adapted and implemented in any programming language.

SwitchEngineListener	Represents a listener for Switch Engine events.

Method Summary	
void	**notifySwitchEngineEvent**(SwitchInstruction SwitchInstruction, int Reason) Method called by the Switch Engine implementation when a Switch Engine event occurs, if the Targeting Engine application has registered as a listener for the event.

SwitchEngineManager	An application may use this object to control the behaviour of the switch engine.

Method Summary	
abstract void	**addInstruction**(SwitchInstruction[] switchInstructions) Add SwitchInstruction(s) to the Switch Engine active list.
static SwitchEngineManager	**getInstance**() gets the singleton instance of the Switch Engine manager for use by a privileged application.
abstract SwitchInstruction[]	**getInstruction**(int[] SwitchIDs, int SourceID) The Switch Engine SHALL return the Switch Instruction(s) matching the passed in SwitchID and SourceID.
abstract void	**removeInstruction**(SwitchInstruction[] switchInstructions) The Switch Engine SHALL remove the passed in Switch Instruction(s) from the Switch Engine active list.
abstract void	**removeSwitchEngineListener**(SwitchEngineListener Listener) Removes the previously added SwitchEngineListener.
abstract void	**setSwitchEngineListener**(SwitchEngineListener Listener, byte[] Filter) Adds the listener for Switch Engine events.

SwitchInstruction	A SwitchInstruction encapsulates the information needed by the switch engine to perform a switch operation.

Constructor Summary
SwitchInstruction() SwitchInstruction constructor.
SwitchInstruction(int SourceID, javax.tv.locator.Locator[] Locator, int SwitchID, byte GroupSize, byte PositionInGroup, int SwitchStartTime, int SwitchEndTime, int SwitchWindowEndTime, int ExpirationTime, boolean TriggerSwitch, int InsertionOptions, boolean InsertionIndivisible, boolean EndOfInsertion) SwitchInstruction constructor.

Method Summary	
`byte[]`	**`getEndOfInsertion`**`()` Retrieve the attribute EndOfInsertion.
`int`	**`getExpirationTime`**`()` Retrieve the attribute ExpirationTime.
`byte`	**`getGroupSize`**`()` Retrieve the attribute GroupSize.
`boolean`	**`getInsertionIndivisible`**`()` Retrieve the attribute InsertionIndivisible.
`int`	**`getInsertionOptions`**`()` Retrieve the attribute InsertionOptions.
`javax.tv.locator.Locator[]`	**`getLocator`**`()` Retrieve the attribute Locator.
`byte`	**`getPositionInGroup`**`()` Retrieve the attribute PositionInGroup.
`int`	**`getSourceID`**`()` Retrieve the attribute SourceID.
`int`	**`getSwitchEndTime`**`()` Retrieve the attribute SwitchEndTime.
`int`	**`getSwitchID`**`()` Retrieve the attribute SwitchID.
`int`	**`getSwitchStartTime`**`()` Retrieve the attribute SwitchStartTime.
`int`	**`getSwitchWindowEndTime`**`()` Retrieve the attribute SwitchWindowEndTime.
`boolean`	**`getTriggerSwitch`**`()` Get attribute TriggerSwitch.
`void`	**`setEndOfInsertion`**`(boolean EndOfInsertion)` Set attribute EndOfInsertion to specified parameter.
`void`	**`setExpirationTime`**`(int ExpirationTime)` Set attribute ExpirationTime to specified parameter.
`void`	**`setGroupSize`**`(byte GroupSize)` Set attribute GroupSize to specified parameter.
`void`	**`setInsertionIndivisible`**`(boolean InsertionIndivisible)` Set attribute InsertionIndivisible to True or False.
`void`	**`setInsertionOptions`**`(int InsertionOptions)` Set attribute InsertionOptions to specified parameter.
`void`	**`setLocator`**`(javax.tv.locator.Locator[] Locator)` Set attribute Locator to specified parameter.
`void`	**`setPositionInGroup`**`(byte PositionInGroup)` Set attribute PositionInGroup to specified parameter.
`void`	**`setSourceID`**`(int SourceID)` Set attribute SourceID to specified parameter.
`void`	**`setSwitchEndTime`**`(int SwitchEndTime)` Set attribute SwitchEndTime to specified parameter.
`void`	**`setSwitchID`**`(int SwitchID)` Set attribute SwitchID to specified parameter.
`void`	**`setSwitchStartTime`**`(int SwitchStartTime)` Set attribute SwitchStartTime to specified parameter.
`void`	**`setSwitchWindowEndTime`**`(int SwitchWindowEndTime)` Set attribute SwitchWindowEndTime to specified parameter.
`void`	**`setTriggerSwitch`**`(boolean TriggerSwitch)` Set attribute TriggerSwitch.

SwitchEngineException	Allows an exception to pass back the SwitchInstruction associated with the Exception.

Constructor Summary	
`SwitchEngineException()`	

Method Summary	
`SwitchInstruction[]`	`SwitchInstructionException()` Returns the SwitchInstruction(s) that caused the exception.

Annex A

OCAP 1.1 Digital Program Insertion API

(This annex forms an integral part of this Recommendation)

This annex defines a Java application programming interface (API) for application command and control of device DPI functionality.

Package org.ocap.dpi

Interface Summary	
SwitchEngineListener	This class represents a listener for Switch Engine events.

Class Summary	
SwitchEngineManager	An application may use this class to control the behaviour of the switch engine.
SwitchInstruction	A SwitchInstruction encapsulates the information needed by the switch engine to perform a switch operation.

Exception Summary	
SwitchEngineException	Derived from Exception and allows an exception to pass back the SwitchInstruction associated with the Exception.

A.1 SwitchEngineException class

org.ocap.dpi
Class SwitchEngineException

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ org.ocap.dpi.SwitchEngineException
```

All Implemented Interfaces:

java.io.Serializable

```
public class SwitchEngineException
extends java.lang.Exception
```

Derived from Exception and allows an exception to pass back the SwitchInstruction associated with the Exception.

Constructor Summary
`SwitchEngineException``()`

Method Summary	
`SwitchInstruction[]`	**`SwitchInstructionException`**`()` Returns the SwitchInstruction(s) that caused the exception.

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

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

Constructor Detail

SwitchEngineException

public **SwitchEngineException**()

Method Detail

SwitchInstructionException

```
public SwitchInstruction[] SwitchInstructionException()
```

Returns the SwitchInstruction(s) that caused the exception.

Returns:

SwitchInstruction[] SwitchInstruction(s) that caused the Exception to be thrown.

A.2 SwitchEngineListener interface

org.ocap.dpi
Interface SwitchEngineListener

All Superinterfaces:

```
    java.util.EventListener
```

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

This class represents a listener for Switch Engine events.

Field Summary	
`static int`	**`COMPONENT_START`** For an L1 Switch a component switch has been completed.
`static int`	**`INSERTION_START`** Switch Instruction was successfully executed and an insertion was accomplished.
`static int`	**`INSTRUCTION_EXPIRED`** Switch Instruction was removed because it has passed its valid time based on its ExpirationTime.
`static int`	**`MISSED_TRIGGER`** The Trigger is position_in_set = 2 or greater and the Switch Instruction's trigger with position_in_set = 1 has not been processed for this Switch set.
`static int`	**`NO_TARGET`** No target for insertion was found at the Locator included in the SwitchInstruction.
`static int`	**`UNARMED_TRIGGER`** A Trigger was detected but no matching SwitchInstruction was found.

Method Summary	
`void`	**`notifySwitchEngineEvent`**`(SwitchInstruction SwitchInstruction, int Reason)` Method called by the Switch Engine implementation when a Switch Engine event occurs, if the Targeting Engine application has registered as a listener for the event.

Field Detail

UNARMED_TRIGGER

```
static final int UNARMED_TRIGGER
```

A Trigger was detected but no matching SwitchInstruction was found.

MISSED_TRIGGER

```
static final int MISSED_TRIGGER
```

The Trigger is position_in_set = 2 or greater and the Switch Instruction's trigger with position_in_set = 1 has not been processed for this Switch set.

NO_TARGET

```
static final int NO_TARGET
```

No target for insertion was found at the Locator included in the SwitchInstruction.

INSTRUCTION_EXPIRED

```
static final int INSTRUCTION_EXPIRED
```

Switch Instruction was removed because it has passed its valid time based on its ExpirationTime. This may not be an error if it is for a different source than has been tuned during the Insertion Group.

INSERTION_START

```
static final int INSERTION_START
```

Switch Instruction was successfully executed and an insertion was accomplished.

COMPONENT_START

```
static final int COMPONENT_START
```

For an L1 Switch a component switch has been completed.

Method Detail

notifySwitchEngineEvent

```
void notifySwitchEngineEvent(SwitchInstruction SwitchInstruction,
                             int Reason)
```

Method called by the Switch Engine implementation when a Switch Engine event occurs, if the Targeting Engine application has registered as a listener for the event.

Parameters:

`SwitchInstruction` – The SwitchInstruction that is associated with this Switch Engine event.

`Reason` – The reason for the event notification.

- UNARMED_TRIGGER – A Trigger was detected but no matching SwitchInstruction was found.
- MISSED_TRIGGER – The Trigger is position_in_set = 2 or greater and the trigger where position_in_set = 1, has not been processed for this Switch Group.
- NO_TARGET – No target for insertion was found at the Locator(s) included in the SwitchInstruction.
- INSTRUCTION_EXPIRED – Switch Instruction was removed because it has passed its valid time based on its ExpirationTime.
- INSERTION_START – Switch has been accomplished.
- COMPONENT_START – For an L1 Switch, a component switch has been completed.

A.3 SwitchEngineManager Class

org.ocap.dpi
Class SwitchEngineManager

```
java.lang.Object
  └ org.ocap.dpi.SwitchEngineManager
```

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

An application may use this class to control the behaviour of the switch engine.

Method Summary	
`abstract void`	**`addInstruction`**`(SwitchInstruction[] switchInstructions)` Add SwitchInstruction(s) to the Switch Engine active list.
`static SwitchEngineManager`	**`getInstance`**`()` gets the singleton instance of the Switch Engine manager for use by a privileged application.
`abstract SwitchInstruction[]`	**`getInstruction`**`(int[] SwitchIDs, int SourceID)` The Switch Engine SHALL return the Switch Instruction(s) matching the passed in SwitchID and SourceID.
`abstract void`	**`removeInstruction`**`(SwitchInstruction[] switchInstructions)` The Switch Engine SHALL remove the passed in Switch Instruction(s) from the Switch Engine active list.
`abstract void`	**`removeSwitchEngineListener`**`(SwitchEngineListener Listener)` Removes the previously added SwitchEngineListener.
`abstract void`	**`setSwitchEngineListener`**`(SwitchEngineListener Listener, byte[] Filter)` Adds the listener for Switch Engine events.

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

Method Detail

getInstance

```
public static SwitchEngineManager getInstance()
```

Gets the singleton instance of the Switch Engine manager for use by a privileged application.

Throws:

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

setSwitchEngineListener

```
public abstract void setSwitchEngineListener(SwitchEngineListener Listener,
                                             byte[] Filter)
```

Adds the listener for Switch Engine events.

Parameters:

`Listener` – Is an instance of a SwitchEngineListener whose SwitchEngineListener method SHALL be called when a Switch Engine event occurs.

`Filter` – A byte array of filters enabling any or all of the following Switch Engine events.

- UNARMED_TRIGGER
- MISSED_TRIGGER
- NO_TARGET
- INSTRUCTION_EXPIRED
- INSERTION_START
- COMPONENT_START

Throws:

`java.lang.IllegalArgumentException` – Listener is null.

removeSwitchEngineListener

```
public abstract void removeSwitchEngineListener(SwitchEngineListener Listener)
```

Removes the previously added SwitchEngineListener.

Parameters:

`Listener` – is the Switch Engine Listener to disable. Does nothing if it was never added, has been removed, or is null.

addInstruction

```
public abstract void addInstruction(SwitchInstruction[] switchInstructions)
```

Add SwitchInstruction(s) to the Switch Engine active list.

If TriggerSwitch is FALSE the Switch Engine SHALL perform an L0 Switch when time is at or past the SwitchStartTime in the SwitchInstruction. For all other SwitchInstructions, the Switch Engine SHALL store the Switch Instruction(s) in its active list. For any Switch Instruction that have an identical SwitchID and SourceID to an already existing Switch Instruction, the Switch Engine SHALL replace the Switch Instruction and a Duplicate Switch Instruction Exception is thrown.

Parameters:

`switchInstructions` –

Throws:

`java.lang.SecurityException` – Calling application does not have MonitorAppPermission("dpi")

`java.lang.IllegalArgumentException` – If the BreakID and SourceID do not identify a SwitchInstruction.

`java.lang.IllegalArgumentException` – Negative SourceID `java.lang.IllegalArgumentException` – ExpirationTime passed

`java.lang.IllegalArgumentException` – Unknown InsertionOptions

`java.lang.IllegalArgumentException` – SwitchEndTime is before SwitchStartTime

`java.lang.IllegalArgumentException` – Duplicate SwitchInstruction

`InvalidLocator` – if Locator is invalid

`InvalidLocator` – Locator is null

`InvalidLocator` – Locator not found

removeInstruction

```
public abstract void removeInstruction(SwitchInstruction[] switchInstructions)
```

The Switch Engine SHALL remove the passed in Switch Instruction(s) from the Switch Engine active list. If any argument is –1, then the Switch Engine SHALL treat it as a "wild card" such that the Switch Engine shall process all Switch Instructions that match that field. (e.g., if SourceID = –1, then remove the Switch Instructions matching the SwitchID and all SourceIDs).

Parameters:

`switchInstructions` – See SwitchInstruction class.

Throws:

`java.lang.IllegalArgumentException` – If the BreakID and SourceID do not identify a SwitchInstruction.

`java.lang.SecurityException` – Calling application does not have MonitorAppPermission("dpi")

getInstruction

```
public abstract SwitchInstruction[] getInstruction(int[] SwitchIDs,
                                                   int SourceID)
```

The Switch Engine SHALL return the Switch Instruction(s) matching the passed in SwitchID and SourceID. If any argument is –1 then the Switch Engine SHALL treat it as a "wild card" such that all Switch Instructions that match that field are processed. (e.g., if SourceID = –1 then get the Switch Instructions matching the SwitchID and all SourceIDs).

Parameters:

`SwitchIDs` – One or more numbers uniquely identifying the SwitchInstructions to return.

`SourceID` – Source Identifier for the SwitchInstructions.

Returns:

SwitchInstruction. An object that is created by the DPI Targeting Engine and passed to the implementation via the Switch engine API. The object SHALL be copied by the method and cannot be changed after loaded.

Throws:

`java.lang.IllegalArgumentException` – If the SwitchID and SourceID do not identify a SwitchInstruction.

`java.lang.SecurityException` – Calling application does not have MonitorAppPermission("dpi")

A.4 SwitchInstruction Class

org.ocap.dpi
Class SwitchInstruction

```
java.lang.Object
  └ org.ocap.dpi.SwitchInstruction
```

```
public class SwitchInstruction
extends java.lang.Object
```

A SwitchInstruction encapsulates the information needed by the switch engine to perform a switch operation. A switch operation is performed according to the SwitchInstruction on receipt of a DPI trigger or when a point on a metadata timeline is reached.

Switches may be trigger-based or time-based. A SwitchInstruction MAY identify a set of triggers, and/or identify a segment of time during which a switch may occur.

A SwitchInstruction object MAY be instantiated by a targeting engine application and passed to the implementation via SwitchEngineManager.addInstruction().

For the attributes of the SwitchInstruction, see the SwitchInstruction constructor parameter list.

Field Summary	
`static int`	**`GO_TO_BLACK`** Go to black at SwitchEndTime even if insert program is not complete.
`static int`	**`PLAY_TO_COMPLETION`** Play inserted program to completion under all circumstances.
`static int`	**`RETURN_TO_PROGRAM`** Return to Program at SwitchEndTime even if inserted content is not complete.

Constructor Summary
`SwitchInstruction``()` SwitchInstruction constructor.
`SwitchInstruction``(int SourceID, javax.tv.locator.Locator[] Locator, int SwitchID, byte GroupSize, byte PositionInGroup, int SwitchStartTime, int SwitchEndTime, int SwitchWindowEndTime, int ExpirationTime, boolean TriggerSwitch, int InsertionOptions, boolean InsertionIndivisible, boolean EndOfInsertion)` SwitchInstruction constructor.

Method Summary	
`byte[]`	**`getEndOfInsertion`**`()` Retrieve the attribute EndOfInsertion.
`int`	**`getExpirationTime`**`()` Retrieve the attribute ExpirationTime.

Method Summary	
byte	**getGroupSize**() Retrieve the attribute GroupSize.
boolean	**getInsertionIndivisible**() Retrieve the attribute InsertionIndivisible.
int	**getInsertionOptions**() Retrieve the attribute InsertionOptions.
javax.tv.locator.Locator[]	**getLocator**() Retrieve the attribute Locator.
byte	**getPositionInGroup**() Retrieve the attribute PositionInGroup.
int	**getSourceID**() Retrieve the attribute SourceID.
int	**getSwitchEndTime**() Retrieve the attribute SwitchEndTime.
int	**getSwitchID**() Retrieve the attribute SwitchID.
int	**getSwitchStartTime**() Retrieve the attribute SwitchStartTime.
int	**getSwitchWindowEndTime**() Retrieve the attribute SwitchWindowEndTime.
boolean	**getTriggerSwitch**() Get attribute TriggerSwitch.
void	**setEndOfInsertion**(boolean EndOfInsertion) Set attribute EndOfInsertion to specified parameter.
void	**setExpirationTime**(int ExpirationTime) Set attribute ExpirationTime to specified parameter.
void	**setGroupSize**(byte GroupSize) Set attribute GroupSize to specified parameter.
void	**setInsertionIndivisible**(boolean InsertionIndivisible) Set attribute InsertionIndivisible to True or False.
void	**setInsertionOptions**(int InsertionOptions) Set attribute InsertionOptions to specified parameter.
void	**setLocator**(javax.tv.locator.Locator[] Locator) Set attribute Locator to specified parameter.
void	**setPositionInGroup**(byte PositionInGroup) Set attribute PositionInGroup to specified parameter.
void	**setSourceID**(int SourceID) Set attribute SourceID to specified parameter.
void	**setSwitchEndTime**(int SwitchEndTime) Set attribute SwitchEndTime to specified parameter.
void	**setSwitchID**(int SwitchID) Set attribute SwitchID to specified parameter.
void	**setSwitchStartTime**(int SwitchStartTime) Set attribute SwitchStartTime to specified parameter.
void	**setSwitchWindowEndTime**(int SwitchWindowEndTime) Set attribute SwitchWindowEndTime to specified parameter.
void	**setTriggerSwitch**(boolean TriggerSwitch) Set attribute TriggerSwitch.

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

Field Detail

GO_TO_BLACK

```
public static final int GO_TO_BLACK
```

Go to black at SwitchEndTime even if insert program is not complete.

RETURN_TO_PROGRAM

```
public static final int RETURN_TO_PROGRAM
```

Return to Program at SwitchEndTime even if inserted content is not complete.

PLAY_TO_COMPLETION

```
public static final int PLAY_TO_COMPLETION
```

Play inserted program to completion under all circumstances.

Constructor Detail

SwitchInstruction

```
public SwitchInstruction()
```

SwitchInstruction constructor. Can be used to Instantiate a SwitchInstruction w/o initializing the attributes.

SwitchInstruction

```
public SwitchInstruction(int SourceID,
                         javax.tv.locator.Locator[] Locator,
                         int SwitchID,
                         byte GroupSize,
                         byte PositionInGroup,
                         int SwitchStartTime,
                         int SwitchEndTime,
                         int SwitchWindowEndTime,
                         int ExpirationTime,
                         boolean TriggerSwitch,
                         int InsertionOptions,
                         boolean InsertionIndivisible,
                         boolean EndOfInsertion)
```

SwitchInstruction constructor. Can be used to initialize all attributes in a SwitchInstruction in place of the set methods.

Parameters:

`SourceID` – The source id of the service on which this instruction may cause a switch.

`Locator` – A Locator that identifies the component or program for insertion.

`SwitchID` – Switch Identifier. Matches the trigger identifier within a set of DPI triggers (synchronized_event_id).

`GroupSize` – Indicates the number of switches within a related group. A group corresponds to an advertising break, which may contain several ad spots.

`PositionInGroup` – Indicates the position of this switch within a group of insertions, zero based.

`SwitchStartTime` – Point on a metadata timeline at which an L0 switch may occur.

`SwitchEndTime` – Point on a metadata timeline at which the insertion segment is over and the switch engine should present the Primary Service.

`SwitchWindowEndTime` – Point on a metadata timeline at which an L0 switch may not occur. In the case where the service is selected at a point past the switchTimeStart, this value indicates the point when the switch should no longer occur.

`ExpirationTime` – The time at which the instruction is no longer valid and SHALL NOT cause a switch to occur. The switch engine MAY remove the instruction at this time. The time is num seconds from receipt by switch engine of instruction.

`InsertionIndivisible` – If true and the first trigger in an Insertion Group is missed then subsequent triggers in the Group SHALL be ignored. Used in conjunction with the GroupSize and PositionInGroup attributes.

`TriggerSwitch` – boolean with the following meanings:

- If TRUE – DPI Trigger initiates each switch.
- If FALSE – Metadata time initiates each switch.

NOTE – If TriggerSwitch = FALSE and time is past the SwitchStartTime when the SwitchInstruction is loaded and executed immediately. This assumes that the switch engine is tuned to a service and has established a valid metadata timeline.

`InsertionOptions` – Options for insertion of content.

- GO_TO_BLACK – Go to black at SwitchEndTime even if insert program is not complete.
- RETURN_TO_PROGRAM – Return to Program at SwitchEndTime even if inserted content is not complete.
- PLAY_TO_COMPLETION – Play inserted program to completion under all circumstances.

`EndOfInsertion` – Identifies if the Switch Instruction should return to the Primary Service. Locator is a don't care.

Throws:

`java.lang.IllegalArgumentException` – Negative SourceID

`java.lang.IllegalArgumentException` – ExpirationTime passed

`java.lang.IllegalArgumentException` – Unknown InsertionOptions

`java.lang.IllegalArgumentException` – SwitchEndTime is before SwitchStartTime

`java.lang.IllegalArgumentException` – Duplicate SwitchID and SourceID.

`InvalidLocator` – Locator is invalid

`InvalidLocator` – Locator is null

`InvalidLocator` – Locator not found

Method Detail

setSwitchID

```
public void setSwitchID(int SwitchID)
```

Set attribute SwitchID to specified parameter.

Parameters:

`SwitchID` – Switch Identifier.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getSwitchID

```
public int getSwitchID()
```

Retrieve the attribute SwitchID.

Returns:

SwitchID

setGroupSize

```
public void setGroupSize(byte GroupSize)
```

Set attribute GroupSize to specified parameter.

Parameters:

`GroupSize` – Number of Switches within a group.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getGroupSize

```
public byte getGroupSize()
```

Retrieve the attribute GroupSize.

Returns:

GroupSize

setPositionInGroup

```
public void setPositionInGroup(byte PositionInGroup)
```

Set attribute PositionInGroup to specified parameter.

Parameters:

`PositionInGroup` – Position of Switch within a group.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getPositionInGroup

```
public byte getPositionInGroup()
```

Retrieve the attribute PositionInGroup.

Returns:

PositionInGroup

setSourceID

```
public void setSourceID(int SourceID)
```

Set attribute SourceID to specified parameter.

Parameters:

`SourceID` – source identifier.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getSourceID

```
public int getSourceID()
```

Retrieve the attribute SourceID.

Returns:

SourceID

setInsertionIndivisible

```
public void setInsertionIndivisible(boolean InsertionIndivisible)
```

Set attribute InsertionIndivisible to True or False.

Parameters:

`InsertionIndivisible` – Indicator of whether a partial Insertion Period can be processed.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getInsertionIndivisible

`public boolean` **`getInsertionIndivisible`**`()`

Retrieve the attribute InsertionIndivisible.

Returns:

InsertionIndivisible

setTriggerSwitch

`public void` **`setTriggerSwitch`**`(boolean TriggerSwitch)`

Set attribute TriggerSwitch.

Parameters:

`TriggerSwitch` – boolean with the following meanings:

- TRUE – DPI Trigger initiated Break
- FALSE – Metadata time initiated Break

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getTriggerSwitch

`public boolean` **`getTriggerSwitch`**`()`

Get attribute TriggerSwitch.

Returns:

TriggerSwitch

setSwitchStartTime

`public void` **`setSwitchStartTime`**`(int SwitchStartTime)`

Set attribute SwitchStartTime to specified parameter.

Parameters:

`SwitchStartTime` – Time insertion is to begin.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getSwitchStartTime

```
public int getSwitchStartTime()
```

Retrieve the attribute SwitchStartTime.

Returns:

SwitchStartTime

setSwitchEndTime

```
public void setSwitchEndTime(int SwitchEndTime)
```

Set attribute SwitchEndTime to specified parameter.

Parameters:

`SwitchEndTime` – End of Insertion Period

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getSwitchEndTime

```
public int getSwitchEndTime()
```

Retrieve the attribute SwitchEndTime.

Returns:

SwitchEndTime

setExpirationTime

```
public void setExpirationTime(int ExpirationTime)
```

Set attribute ExpirationTime to specified parameter.

Parameters:

`ExpirationTime` – The time when the Switch Instruction is no longer valid for an Insertion and can be removed.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getExpirationTime

`public int `**`getExpirationTime`**`()`

Retrieve the attribute ExpirationTime.

Returns:

ExpirationTime

setSwitchWindowEndTime

`public void `**`setSwitchWindowEndTime`**`(int SwitchWindowEndTime)`

Set attribute SwitchWindowEndTime to specified parameter.

Parameters:

`SwitchWindowEndTime` – point on a metadata timeline at which an L0 switch may not occur. In the case where the service is selected at a point past the switchStartTime, this value indicates the point when the switch should no longer occur.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getSwitchWindowEndTime

`public int `**`getSwitchWindowEndTime`**`()`

Retrieve the attribute SwitchWindowEndTime.

Returns:

SwitchWindowEndTime

setInsertionOptions

`public void `**`setInsertionOptions`**`(int InsertionOptions)`

Set attribute InsertionOptions to specified parameter.

Parameters:

`InsertionOptions` – Options for insertion of content.

- GO_TO_BLACK – Go to BLACK at SwitchEndTime even if insert program is not complete.
- RETURN_TO_PROGRAM – Return to Program at SwitchEndTime even if insert program is not complete.
- PLAY_TO_COMPLETION – Play inserted program to completion under all circumstances.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getInsertionOptions

`public int `**`getInsertionOptions`**`()`

Retrieve the attribute InsertionOptions.

Returns:

InsertionOptions

setLocator

`public void `**`setLocator`**`(javax.tv.locator.Locator[] Locator)`

Set attribute Locator to specified parameter.

Parameters:

`Locator` – Identifies the component or program for insertion. Locator pointing to media content to switch to.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getLocator

`public javax.tv.locator.Locator[] `**`getLocator`**`()`

Retrieve the attribute Locator.

Returns:

javax.tv.locator.Locator[]

setEndOfInsertion

`public void `**`setEndOfInsertion`**`(boolean EndOfInsertion)`

Set attribute EndOfInsertion to specified parameter.

Parameters:

`EndOfInsertion` – Identifies this as an end of Insertion Swtich Instruction that returns to the primary service.

Throws:

`java.lang.IllegalStateException` – this method is called when the SwitchInstruction is Active (Has been added with the addInstruction).

getEndOfInsertion

```
public byte[] getEndOfInsertion()
```

Retrieve the attribute EndOfInsertion.

Returns:

EndOfInsertion

Printed in Switzerland
Geneva, 2010