

OpenCable™ Specifications OCAP Extensions

OCAP Multiscreen Manager (MSM) Extension

OC-SP-OCAP-MSM-I01-071012

ISSUED

Notice

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

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

© Copyright 2007 Cable Television Laboratories, Inc.
All rights reserved.

Document Status Sheet

Document Control Number:	OC-SP-OCAP-MSM-I01-071012			
Document Title:	OCAP Multiscreen Manager (MSM) Extension			
Revision History:	I01 – Released 10/12/07			
Date:	October 12, 2007			
Status:	Work in Progress	Draft	Issued	Closed
Distribution Restrictions:	Author Only	CL/Member	CL/Member/ Vendor	Public

Key to Document Status Codes:

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

Trademarks:

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

Contents

1	SCOPE	1
1.1	Introduction and Purpose.....	1
1.2	Requirements.....	1
2	REFERENCES	2
2.1	Normative References.....	2
2.2	Informative References.....	2
2.3	Reference Acquisition.....	2
3	TERMS AND DEFINITIONS	3
4	ABBREVIATIONS AND ACRONYMS	5
5	OVERVIEW	6
6	SIGNALING	7
6.1	Multiscreen Usage Descriptor.....	7
7	APPLICATION PROGRAM INTERFACE	9
7.1	Multiscreen Manager.....	9
7.2	Screens.....	9
7.2.1	<i>Display Screens</i>	9
7.2.2	<i>Logical Screens</i>	10
7.2.3	<i>Screen Categories</i>	12
7.2.4	<i>Screen Identifiers</i>	12
7.3	Multiscreen Configurations.....	12
7.3.1	<i>Multiscreen Configuration Types</i>	13
7.3.2	<i>Default Service Context Association Screen</i>	13
7.3.3	<i>Default Multiscreen Configuration</i>	13
7.3.4	<i>Changing Multiscreen Configurations</i>	14
7.4	Baseline HAVi Model Extensions.....	19
7.4.1	<i>HScreen</i>	19
7.4.2	<i>HScreenDevice</i>	22
7.4.3	<i>HScreenConfiguration</i>	24
7.4.4	<i>HScreenConfigTemplate</i>	25
7.4.5	<i>HVideoDevice</i>	26
7.5	Security.....	27
7.6	System Property.....	27
7.7	Registry of Constants.....	28
ANNEX A	MULTISCREEN MANAGER API	30
	Package org.ocap.ui.....	30
	org.ocap.ui Interface MultiScreenConfigurableContext.....	31
	org.ocap.ui Interface MultiScreenConfiguration.....	50
	org.ocap.ui Interface MultiScreenContext.....	59
	org.ocap.ui Class MultiScreenManager.....	70
	Package org.ocap.ui.event.....	88
	org.ocap.ui.event Class MultiScreenConfigurationEvent.....	89
	org.ocap.ui.event Interface MultiScreenConfigurationListener.....	93
	org.ocap.ui.event Class MultiScreenContextEvent.....	94
	org.ocap.ui.event Interface MultiScreenContextListener.....	99

org.ocap.ui.event Class MultiScreenEvent100
org.ocap.ui.event Class MultiScreenResourceEvent102
ANNEX B MULTISCREEN RECONFIGURATION CONSTRAINTS105

Tables

Table 6-1 - Multiple Screen Usage Descriptor7
Table 7-1 - Additional Monitor Application Permissions for Multiscreen Manager Extension27

1 SCOPE

1.1 Introduction and Purpose

This document defines a modular extension to the OpenCable Application Platform [OCAP] Profile for OCAP Host devices that support Multiscreen Management functionality as defined by this specification. Multiscreen Management functionality is defined as a standardized software interface for allowing interoperable OCAP applications to effectively use the resources of a Host device that supports multiple screens. The OCAP Profile and this OCAP MSM Extension were developed by Cable Television Laboratories, Inc. (CableLabs), in conjunction with representatives from a number of its member cable operating companies, as well as leading software firms.

The OCAP MSM Extension is based on the OCAP 1.0 and subsequent profiles.

The OCAP MSM Extension is an extension of the OCAP Profile that includes all required Application Program Interfaces (APIs), content and data formats, and protocols, up to the application level. Applications developed to the OCAP MSM Extension will be executed on OpenCable-compliant Host devices. The OCAP MSM Extension allows cable operators to deploy interoperable applications to manage multiple screen functionality on OpenCable-compliant Host devices connected to their networks. This profile allows cable operators to have a standardized interface for managing multiscreen functionality and its state, such as PiP (picture-in-picture) and PoP (picture-outside-picture) screens across multiple Host device vendors.

The OCAP MSM Extension is applicable to a wide variety of hardware and operating systems to allow Consumer Electronics (CE) manufacturers flexibility in implementation. A primary objective in defining the OCAP MSM Extension is to enable competing implementations by CE manufacturers while maintaining a consistent and interoperable programmatic interface for use by cable operator-defined applications as well as cable network-independent applications that wish to be aware of multiscreen functionality.

1.2 Requirements

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

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

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

[OCAP] OpenCable Application Platform Specification, Profile 1.0; OC-SP-OCAP1.0.1-070824, August 24, 2007, Cable Television Laboratories, Inc.

2.2 Informative References

None.

2.3 Reference Acquisition

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

3 TERMS AND DEFINITIONS

This specification uses the following terms:

Audio Focus Screen	A screen whose audio sources are selected for rendition on any audio presentation device associated with any enabled video output ports with which the screen is associated directly (in case it is a display screen) or indirectly (in case it is a mapped logical screen).
Display Mapper	A logical process of compositing the normalized coordinate space of a logical screen to the normalized coordinate space of a display screen (or some portion thereof).
Display Multiscreen Configuration	A multiscreen configuration consisting only of display screens. Every per-platform multiscreen configuration is a display multiscreen configuration. A display multiscreen configuration's configuration type is SCREEN_CONFIGURATION_DISPLAY.
Display Screen	A type of screen that models a final output composition of a physical, display device. A display screen is associated with one or more video output ports that are themselves either (1) attached to an internal, integrated display or (2) attached to a physical output port cable of being connected to an external display device via some well-defined interface, e.g., HDMI, IEEE 1394, etc.
General Screen	A logical screen that is assigned to the <i>General</i> screen category. A General screen is typically capable of being reconfigured so that it may serve different use cases. For example, it may be configured as a PiP screen at some time, but as a PoP screen at another time.
Logical Screen	A type of screen that is not directly associated with a physical display device but may be mapped through a display mapper to a display screen. A logical screen is either mapped or unmapped. If mapped, it is associated with a display screen; if unmapped, it is not associated with a display screen. In exceptional cases, a logical screen may be directly associated with a video output port, in which case it is simultaneously serving as a logical screen and as a quasi-display screen.
Main Screen	A display screen or a logical screen that is assigned to the <i>Main</i> screen category. A Main screen is always coextensive with to the full screen extent of a display device (unless further mapped by the display device itself without the knowledge of MSM).
Multiscreen Configurable Context	An extension of a multiscreen context that provides the ability to configure (mutate) state information related to multiscreen management functionality.
Multiscreen Configuration	An identifiable collection of screens that are presenting or able to present content, where this collection may (but need not) be a proper subset of all accessible screens on a given platform implementation or on a given display screen. At any given time, a single multiscreen configuration applies to both (1) the platform as a whole, in which case the screens in the configuration are display screens, and (2) each display screen, in which case the screens in the configuration are either the display screen itself or a set of logical screens mapped to that display screen. A given underlying screen may be present in multiple multiscreen configurations at a given time.
Multiscreen Configuration Type	A characterization of a multiscreen configuration based primarily upon its constituent screen categories. The set of multiscreen configuration types is further divided into (1) the singleton display multiscreen configuration type, known as a display multiscreen configuration, and (2) all other multiscreen configuration types, known as non-display multiscreen configurations.

Multiscreen Context	A set of state information that extends a HAVi Screen in order to permit interoperation in a platform that implements multiple screens. The baseline multiscreen context provides functionality to discover (but not mutate) this state information.
Multiscreen Manager	A singleton management component provided by a platform implementation that, through the features defined in this specification, enables effective use of multiple screens. Also referred to as <i>multiple screen manager</i> .
Overlay Screen	A logical screen that is assigned to the <i>Overlay</i> screen category. The z-order of an Overlay screen places it more front-most than all other non-overlay screens.
Per-Display Multiscreen Configuration	A non-display multiscreen configuration that defines or may be used to define the active or potentially active set of screens associated with a given display screen. Each underlying display screen in the currently active per-platform multiscreen configuration contains a state variable to define its active per-display multiscreen configuration.
Per-Platform Multiscreen Configuration	A display multiscreen configuration that currently defines or may be used to define the active or potentially active set of display screens. The currently active per-platform multiscreen configuration is a state variable of the multiscreen manager singleton instance.
PiP Screen	A logical screen that is assigned to the <i>PiP</i> screen category. A PiP screen typically intersects with a Main screen, and appears more front-most (in z-order) than this Main screen.
PoP Screen	A logical screen that is assigned to the <i>PoP</i> screen category. A PoP screen is typically tiled along with one or more other PoP screens so as to fill the extent of a display screen.
Screen	A set of (logical and possibly physical) resources that enable presentation of some combination of a background color, a background image, video, and graphics. A screen is typed as either a display screen or a logical screen.
Screen Category	A grouping of screens according to their usage or configurability characteristics.
Screen Identifier	A platform-unique designation of an underlying screen and its respective resources that permits discrimination between screens of different categories or screens within a single category.
Screen Type	A top-level division of screens into two types: display screens and logical screens.

4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations:

AIT	Application Information Table
HAVi	Home Audio-Visual Interface
MSM	Multiscreen Manager
PiP	Picture-in-Picture
PoP	Picture-outside-Picture
XAIT	Extended Application Information Table

5 OVERVIEW

The OCAP Multiscreen Manager (MSM) Extension defines a collection of functionality and behaviors to permit the effective use of multiple displays and multiple logical screens on platform implementations that support these features.

The degree to which a specific platform implementation supports this extension is determined by the platform and device manufacturer or by other, external profiles of this extension, and is typically dictated by the type and configurability of hardware, including the number of available tuners or independent input sources, the number of available video decoding pipelines, the number of integrated displays, the number of independent video (and audio) output ports, the amount of memory available for video and graphics buffering, and other unspecified resources.

This extension takes the form of:

- Signaling Mechanisms (descriptors)
- Application Program Interfaces (APIs)
- Platform Behavior (semantics)
- Application Usage Constraints

In general, the APIs defined by this extension take the form of extensions to existing functionality and APIs defined by the HAVi User Interface as defined by the package `org.havi.ui`. In particular, much of the functionality defined herein takes the form of interfaces implemented by the concrete class used by a platform to support the `org.havi.ui.HScreen` class.

6 SIGNALING

The OCAP MSM Extension defines an optional descriptor used to signal MSM-related behavior for an application. In the absence of such a descriptor, default semantics are defined.

6.1 Multiscreen Usage Descriptor

The OCAP MSM Extension extends the OCAP Profile [OCAP] through the multiple screen usage descriptor as defined below. An OCAP application signaled in an AIT or in an XAIT MAY include one, but no more than one, multiple screen usage descriptor in the application descriptor loop that applies to the application.

If no multiple screen usage descriptor is present in the application descriptor loop of some application, then an OCAP Host device SHALL consider this to be equivalent to the presence of a multiple screen usage descriptor whose `terminate_on_condition` flag is clear ('0'), whose `disallow_partial_display_graphics_mapping` flag is clear ('0'), and whose `allow_default_device_reconfig` flag is clear ('0').

Note: The effect produced by the above is such that, by default, (1) an application's default graphics plane can be scaled (along with its default background and video planes) to a portion of a display screen through the means of an intermediate logical screen, (2) an application's default screen devices cannot be arbitrarily reconfigured or unresourced, and (3) if this last condition would necessarily hold, then the application is suspended, or, if suspension is not possible, terminated.

Table 6-1 - Multiple Screen Usage Descriptor

	No. of bits	Syntax	Value
<code>multiple_screen_usage_descriptor() {</code>			
<code>descriptor_tag</code>	8	uimsbf	0x6E
<code>descriptor_length</code>	8	uimsbf	
<code>terminate_on_condition</code>	1	uimsbf	
<code>disallow_partial_display_graphics_mapping</code>	1	uimsbf	
<code>allow_default_device_reconfig</code>	1	uimsbf	
<code>reserved</code>	5	uimsbf	
<code>}</code>			

descriptor_tag: An 8-bit unsigned integer with the value 0x6E identifies this descriptor.

descriptor_length: An 8-bit unsigned integer that specifies the number of bytes immediately following this field. For this version of this specification, exactly one byte SHALL follow this field.

terminate_on_condition: A 1-bit flag indicating the desired effect on this application's lifecycle in the case that either `disallow_partial_display_graphics_mapping` is set ('1') or `allow_default_device_reconfig` is clear ('0') and the condition specified by one of these two respective flags requires application suspension or termination. If this flag is set ('1'), then the OCAP platform SHALL terminate (destroy) the application on the onset of the governing condition; otherwise, if this flag is clear ('0'), then the OCAP platform SHALL attempt to suspend the application, and, if unable to suspend, SHALL terminate the application on the onset of the governing condition. If an application is suspended as a result of this flag being set,

then when no governing condition obtains, the application SHOULD be resumed, but, if resumption is not possible, e.g., due to insufficient resource availability, SHALL be terminated (destroyed).

In the present context, the term *suspend* is to be interpreted as equivalent to invoking the `pauseXlet()` method of the affected application's initial `Xlet` instance.

disallow_partial_display_graphics_mapping: A 1-bit flag indicating that the application cannot continue to run when a multiscreen configuration changes or service contexts are swapped or moved between screens such that the application's default graphics plane (screen device) would be scaled to operate in a logical screen whose screen area is not mapped to full display screen extent. If set ('1'), then the application SHALL be suspended or terminated on the onset of this condition according to the value of the `terminate_on_condition` flag above; otherwise, if clear ('0'), then the application is not suspended or terminated on the onset of this condition.

allow_default_device_reconfig: A 1-bit flag indicating that the application can continue to run when a multiscreen configuration changes or service contexts are swapped or moved between screens such that (1) one or more of the following screen device configuration (`HScreenConfiguration`) parameters of some default screen device would change: screen area, pixel resolution, or pixel aspect ratio; or (2) the underlying resources of one or more default screen devices that were present before the configuration change or service context swap/move are no longer available after the configuration change or service context swap/move. If clear ('0'), then the application SHALL be suspended or terminated on the onset of this condition according to the value of the `terminate_on_condition` flag above; otherwise, if set ('1'), then the application is not suspended or terminated on the onset of this condition.

Note: An application that signals this flag in the set ('1') state is expected to register listeners for and process both `org.havi.ui.HScreenConfigurationEvent` and `org.ocap.ui.MultiScreenEvent` events.

reserved: This field is reserved for future standardization and shall have a value consisting of all '1's.

7 APPLICATION PROGRAM INTERFACE

The OCAP MSM Extension defines a collection of functionality and behaviors to permit the effective use of multiple displays and multiple logical screens on platform implementations that support these features. The MSM Extension extends the OCAP Profile [OCAP] through the addition of the Multiple Screen Management functionality specified below.

OCAP Host devices that support the OCAP MSM Extension SHALL implement the Java types and their associated semantics as specified by Annex A, as well as the semantics as specified by the following sub-sections.

7.1 Multiscreen Manager

The default `HScreen` for an OCAP application, i.e., the value returned by `HScreen.getDefaultHScreen()`, is an `HScreen` instance that represents the currently active set of default, underlying screen devices and their currently active HAVi screen configurations.

Note: See Section 7.4.1.1, Default HScreen Assignment and `MultiScreenManager.getDefaultHScreen()` for further information on an application's default `HScreen`.

An OCAP Host device that implements the `MultiScreenManager` class SHALL provide a distinct logical `HScreen` instance for each secondary and primary viewing screen that is exposed to the OCAP implementation, i.e., both PiP and non-PiP viewing screens.

7.2 Screens

The HAVi User Interface sub-system (referred to as the *baseline HAVi model*) defines the `HScreen` class to be a description of "the final output composition of a device", and indicates that "a platform with two independent displays would support two instances of this class." With the introduction of the MSM Extension, this definition is extended to introduce a distinction between a *display screen* and a *logical screen*.

7.2.1 Display Screens

A screen that models a "final output composition of a [physical, display] device" is referred to by MSM as a *display screen*. In contrast, a screen that is mapped to a display screen through a logical coordination system transformation process, either through hardware or software, is referred to as a *logical screen*.

7.2.1.1 Video Output Port Association

A display screen is associated with (mapped to) zero or more video output ports, which can be individually enabled or disabled as defined by `VideoOutputPort`. If a display screen is not associated with any video output port, then any presentation that is active in the context of the display screen is not rendered by a presentation device.

7.2.1.2 Audio Output Port Association

In the context of MSM, audio outputs and their association with a display screen for the purpose of rendering audio content are considered to be implied by the video output port associations of the display screen.

7.2.1.3 Audio Focus

A display screen designates itself or one of the logical screens that map to it to be its *audio focus* screen. This screen is used to determine which audio sources are rendered on any (implied) audio outputs associated with the video

output ports to which the display screen is mapped. This distinction is required only when there are multiple logical screens that are simultaneously mapped to and presenting on a display screen, such as would be the case in either a PiP or PoP multiscreen configuration.

If no logical screen is mapped to a display screen, or audio focus is not assigned to such a logical screen, then audio focus is assigned to the display screen itself. In this case, any `HScreenDevice` instances associated with the display screen become potential candidates as audio sources.

7.2.2 Logical Screens

A *logical screen* is a natural extension of the baseline HAVi model of an `HScreen` where one reads "device" in the phrase "the final output composition of a device" as a "logical device" as opposed to a physical device. This extension is not dissimilar to the commonly understood notion of a *logical* disk drive, where a portion of a physical disk drive is viewed as if it were an independent disk drive in its own right.

From the perspective of an OCAP application, a *logical screen* is for all intents and purposes the same as a *display screen*. It possesses all of the standard attributes and behaviors of an existing (non-multiscreen) `HScreen` instance: a set of default screen background, video, and graphics screen devices, an optional set of non-default screen devices of these same types, one or more sets of coherent screen device configurations, an ability to atomically modify a set of screen devices in order to establish a coherent set of screen configurations for these devices, and a mechanism for determining the best configuration of a screen device of a specific type given a screen device configuration template.

Each *logical screen* possesses its own, independent, normalized screen coordinate space as defined by the baseline HAVi model. A *logical screen* is (typically) associated with a *display screen* and a *display area* on that display screen. Through this association, the normalized screen coordinate space of a logical screen is mapped to a screen rectangle in the normalized screen coordinate space of the display screen. This screen rectangle (hereafter referred to as *display area*) may coincide with the entire display screen, in which case the mapping is an identity mapping between the two normalized screen coordinate spaces, or it may coincide with a portion of the display screen. In this latter case, the display area may be wholly contained within the visible region of the display screen, it may be wholly outside the visible region of the display screen, or it may intersect with both visible and non-visible (external) parts of the display screen. If some portion of the display area extends outside the visible display region of the display screen (i.e., outside screen rectangle [0,0];[1,1] of the display screen), then it is an implementation option whether or not externally areas are clipped to the (nominally) visible region of the display screen.

Each *logical screen* possesses a set of default screen devices, each represented as an `HScreenDevice` instance of a specific sub-type: `HBackgroundDevice`, `HVideoDevice`, or `HGraphicsDevice`. In addition, like the baseline HAVi model, each logical screen can possess a set of additional non-default screen devices of each of these sub-types.

As with the baseline HAVi model, each underlying screen device resource utilized by a *logical screen* is referenced through a standard sub-type of the `HScreenDevice` class, and each of these screen devices possesses its own set of configuration parameters as defined by the baseline HAVi model and represented by means of a standard sub-type of the `HScreenConfiguration` class. For example, one such parameter is modeled by the template preference `HScreenConfiguration.PIXEL_RESOLUTION`, and is accessible using the helper function `HScreenDevice.getPixelResolution()`. As is the case with `HScreenDevice` instances in a non-multiscreen implementation of the baseline HAVi model, this parameter is also present and accessible through the same methods when referencing a screen device of a *logical screen*. However, in the case of a logical screen as defined here, the pixel resolution refers to the resolution of the screen device in the context of the normalized coordinate space of the *logical screen*, and not the *display screen* to which this logical screen is mapped. Ultimately, the pixel resolution of a screen device of a logical screen does map, by means of a second order coordinate space transformation, to the normalized screen coordinate space of a display screen. However, an application that is using the logical screen need not be aware of this second order transformation, since from a logical processing perspective, the logical screen is no different from a final output signal to a real, physical display device.

The mapping between a logical screen's normalized coordinate space and the associated display area in the display screen's normalized coordinate space (the second order coordinate space transformation cited above), referred to hereafter as a *display mapping*, is defined as a *logical mapping* process, which is to say that an MSM implementation need not dedicate a specific hardware component to this mapping process. For example, one implementation choice is to (internally) model the union of all sets of (resource-assigned) screen devices of all logical screens that map to a specific display screen as a single set of screen devices in that single display screen, where certain constraints regarding pixel coherency and z-order among this set of screen devices apply. In particular, if some set of pixel coherency rules apply to a given (sub)set of screen devices in some logical screen, then those pixel coherency rules also apply when those screen devices are considered as part of the larger set of (notionally mapped) screen devices in the display screen. Similarly, the z-order of the (notionally mapped) screen devices in the display screen must respect (1) the z-ordering of screen devices within each logical screen, and (2) the z-ordering between logical screens. In other words, the (notionally mapped) screen devices of the different logical screens are not interleaved.

Whether the display mapping process (from a logical to a display screen) is performed by logical or physical (hardware) means, the entity that effects this process is referred to as a *display mapper* for a given *<logical screen, display screen>* pair.

7.2.2.1 Video Output Port Association

Under typical circumstances, a logical screen is indirectly associated with a video output port by virtue of being mapped to a display screen where the display screen is directly associated with a video output port. However, in order to permit more flexibility, MSM allows a logical screen to be directly associated with a video output port.

A logical screen may be directly associated with a video output port in two circumstances: (1) when the logical screen is orphaned, i.e., is not mapped to any display screen but is independently operating as if it were a display screen (in which case it can be directly associated with a video output port), and (2) when the logical screen is parented to a display screen and thus is indirectly associated with the display screen's video output port associations, and, at the same time, is independently operating as if it were a display screen and is additionally directly associated with another video output port. Use cases that embody these two circumstances are described as follows, respectively:

1. An orphaned logical screen is attached to a service context whose content is being recorded, e.g., by DVR functionality, and it is desired to simultaneously output this content to an auxiliary video output port, e.g., a 1394 or an NTSC composite port.
2. A logical screen is presenting as a PiP screen mapped to a display screen in a PiP multiscreen configuration (wherein a Main screen is also simultaneously presenting) and this display screen is associated with the primary video output port; in addition, it is desired that this PiP logical screen be output directly to an auxiliary video output port, e.g., a 1394 or an NTSC composite port.

7.2.2.2 Audio Output Port Association

Logical screens are indirectly associated with audio output ports either through (1) indirect association with a video output port through being mapped to a display screen, or (2) direct association with a video output port as described above.

When multiple logical screens are mapped to a display screen, only one of the logical screens typically contributes audio sources for rendering at a given time. This one logical screen is designated as the *audio focus* screen of a display screen, and is determined by using `MultiScreenConfigurableContext.assignAudioFocus()`.

Because a single logical screen may have multiple sources of audio, as derived from both its default `HScreenDevice` instances as well as non-default instances, it is desirable to control which of these instances

contributes to the logical screen's combined audio output. These sources are determined by using the `MultiScreenConfigurableContext.{add,remove}AudioSources(...)` methods.

7.2.3 Screen Categories

Each screen in a multiscreen configuration is assigned to one of the following pre-defined screen categories or a platform-dependent category whose string representation starts with "x-":

- None (`SCREEN_CATEGORY_NONE`)
- Display (`SCREEN_CATEGORY_DISPLAY`)
- Main (`SCREEN_CATEGORY_MAIN`)
- PiP (`SCREEN_CATEGORY_PIP`)
- PoP (`SCREEN_CATEGORY_POP`)
- Overlay (`SCREEN_CATEGORY_OVERLAY`)
- General (`SCREEN_CATEGORY_GENERAL`)

These screen categories are defined formally as an extensible String enumeration by the above specified fields of `MultiScreenConfiguration`. A screen is categorized as `SCREEN_CATEGORY_NONE` only if no more specific category applies.

7.2.4 Screen Identifiers

Each identifiable set of screen resources that collectively represents a display or logical screen is assigned a unique (string) identifier, where the scope of uniqueness includes all screens accessible through all active multiscreen configurations at a given time. An implementation of MSM MAY enlarge this scope of uniqueness to include all non-active multiscreen configurations at a given time, and furthermore, MAY enlarge the scope to include all continuous time durations over which the platform operates (between cold boot cycles).

An MSM implementation that supports dynamic, run-time addition of display screens or of logical screens SHALL be prepared to assign identifiers, at the time of addition, that maintain the above constraints on minimum scope of uniqueness.

7.3 Multiscreen Configurations

At any given time, a specific (usually, but not necessarily proper) subset of display screens is active in the context of an MSM implementation in the sense that these display screens, if marked as visible, are simultaneously presenting content or are able to present content from one or more logical screens. Such a subset of display screens is referred to as a *per-platform display multiscreen configuration* and is modeled by MSM using an object that implements the `MultiScreenConfiguration` interface and where the `getConfigurationType()` method of this interface returns `MultiScreenConfiguration.SCREEN_CONFIGURATION_DISPLAY`.

The currently active per-platform multiscreen configuration may be obtained with `MultiScreenManager.getMultiScreenConfiguration()` and established with `MultiScreenManager.setMultiScreenConfiguration()`.

In addition, for each display screen that participates in a platform multiscreen configuration, there exists a specific (usually proper) subset of (extant) logical screens that is active in the context of that display screen such that, if some such logical screen is marked as visible, then it is simultaneously presenting content or is able to present content from one or more service contexts or media players. Such a subset of logical screens is referred to as a *per-display multiscreen configuration* and is modeled by MSM using an object that implements the

`MultiScreenConfiguration` interface and where the `getConfigurationType()` method of this interface returns a value other than `MultiScreenConfiguration.SCREEN_CONFIGURATION_DISPLAY`.

The currently active per-display multiscreen configuration of a specific display screen may be obtained with the `MultiScreenContext.getMultiScreenConfiguration()` method and established with `MultiScreenConfigurableContext.setMultiScreenConfiguration()`. The set of all accessible usable per-display multiscreen configurations (which may be used with a specific display screen) may be obtained with `MultiScreenContext.getMultiScreenConfigurations()`.

The set of all accessible multiscreen configurations (whether presently in use or not in use, and whether a per-platform or per-display multiscreen configuration) may be obtained with `MultiScreenManager.getMultiScreenConfigurations()`.

For each non-terminated OCAP application, *A*, the underlying resources of exactly one `HScreen` instance of some currently active multiscreen configuration of a display screen of the currently active per-platform multiscreen configuration SHALL be the same as (equivalent to) the underlying resources of the default screen of *A*.

7.3.1 Multiscreen Configuration Types

Each set of screens that is modeled as a multiscreen configuration is assigned one of the following pre-defined configuration types or a platform-dependent configuration type whose string representation starts with "x-":

- Display (`SCREEN_CONFIGURATION_DISPLAY`)
- Non-PiP (`SCREEN_CONFIGURATION_NON_PIP`)
- PiP (`SCREEN_CONFIGURATION_PIP`)
- PoP (`SCREEN_CONFIGURATION_POP`)
- General (`SCREEN_CONFIGURATION_GENERAL`)

These configuration types are defined formally as an extensible String enumeration by the above specified fields of `MultiScreenConfiguration`.

7.3.2 Default Service Context Association Screen

A multiscreen configuration designates one of its accessible screens to be the *default service context association* screen. This screen is used to determine the default association between a service context and a screen in the absence of more specific association information.

Each pre-defined multiscreen configuration type defines a specific screen to be its *initial* default service context association screen. Subsequent to platform boot time, the default service context association screen of a multiscreen configuration may be changed by using `MultiScreenConfiguration.setDefaultServiceContextScreen(...)`. The current default service context association screen of a multiscreen configuration may be obtained by using `MultiScreenConfiguration.getDefaultServiceContextScreen()`.

Note: For more information, see `MultiScreenManager.setMultiScreenConfiguration(...)` and `MultiScreenConfigurableContext.setMultiScreenConfiguration(...)`, specifically, the description of the *serviceContextAssociations* parameter of these methods.

7.3.3 Default Multiscreen Configuration

After performing a cold boot (or restart) of an OCAP Host Device that implements the MSM Extension, the default (initially active) per-platform multiscreen configuration SHALL be the same multiscreen configuration that was active prior to cold restart, or, if no prior per-platform multiscreen configuration is known, then it SHALL be the

first multiscreen configuration returned by `MultiScreenManager.getMultiScreenConfigurations (SCREEN_CONFIGURATION_DISPLAY)` that is associated with a display screen, which, in turn, is associated with a per-display multiscreen configuration of configuration type `SCREEN_CONFIGURATION_NON_PIP`.

Similarly, the default (initially active) per-display multiscreen configuration of each display screen of the default per-platform multiscreen configuration SHALL be the same multiscreen configuration that was active prior to cold restart, or, if no prior per-display multiscreen configuration is known for some display screen, then it SHALL be the first multiscreen configuration of configuration type `SCREEN_CONFIGURATION_NON_PIP` that is returned by `MultiScreenContext.getMultiScreenConfigurations (SCREEN_CONFIGURATION_NON_PIP)` on that display screen.

When performing any other type of (non-cold) restart, reboot, or reset of an OCAP Host Device or OCAP environment that implements the MSM Extension, then the default (initially active) multiscreen configuration SHALL be the same multiscreen configuration that was active prior to non-cold restart, reboot, or environment reset.

7.3.4 Changing Multiscreen Configurations

A current multiscreen configuration MAY be changed by a privileged application (to which `MonitorAppPermission ("multiscreen.configuration")` has been granted) by means of the `setMultiScreenConfiguration (...)` or `requestMultiScreenConfigurationChange (...)` method of either the `MultiScreenManager` instance or the `MultiScreenConfigurableContext` interface of some display screen. In addition, it MAY be changed by the Host device itself as a result of other events not defined by this specification; for example, a manufacturer could provide an explicit remote control key that causes a specific multiscreen configuration to be activated.

Regardless of what causes the current multiscreen configuration to be changed, the following constraints SHALL apply:

1. Prior to the change, the preconditions specified below apply;
2. During the change, the change processing steps specified below apply, during which time the dynamic state invariants specified below apply;
3. Subsequent to the change, the postconditions specified below apply.

For the purpose of enhanced interoperability, a number of these preconditions and postconditions are expressed in pseudo-code form in the form of assertions in Annex B. The constraints expressed therein are intended to be consistent with the descriptive text that appears in the following sub-sections. For avoidance of doubt, the following descriptive text takes priority over Annex B in case of a discrepancy or partial coverage by the coded form. Notwithstanding this prioritization, Annex B SHALL apply for the purpose of determining conformance of an interoperable implementation of the MSM Extension.

7.3.4.1 Preconditions

Prior to a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING` event that is generated as a result of a multiscreen configuration change, the following ordered preconditions SHALL hold:

1. The Quiescent State Invariants defined in Section 7.3.4.4 below must be satisfied.

7.3.4.2 Change Processing Steps

The MSM implementation SHALL perform the following ordered steps in order to effect a change to the current multiscreen configuration:

1. Generate a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING` event for application dispatching;
2. Starting at this time and until the time that `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED` is generated, do not permit an OCAP application to (1) change the current multiscreen configuration, (2) reserve an `HScreen` or an `HScreenDevice` and their underlying resources, or (3) perform a screen device configuration change or a multiscreen screen context state change;
3. For each OCAP application that has been granted `MonitorAppPermission("multiscreen.-configuration")`, invoke the `notify()` method of any registered `MultiScreenConfigurationListener`, providing as an argument the `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING` generated above;

All such registered listeners SHALL be invoked prior to continuing to the next step; however, an MSM implementation is not required to wait for any or all listeners to return from the `notify()` method prior to continuing.

4. For any `HScreen` instance whose `MultiScreenContext` accessible state will or may change as a result of this change to the current multiscreen configuration, then, if the `HScreen` instance is currently reserved by some OCAP application other than the application that invoked the `setMultiScreenConfiguration(...)` method or reserved by any OCAP application in case this change is a result of a Host device-initiated change (without explicit invocation of this method by an OCAP application), then, for each `HScreenDevice` referenced by the reserved `HScreen`, and, while deferring the notification of any generated `HScreenDeviceReleasedEvent` instances, invoke the `releaseDevice()` method and, if necessary, invoke the `release()` method of the `ResourceClient` that holds the reservation;
5. For any `HScreenDevice` instance whose `HScreenConfiguration` accessible state will or may change as a result of this change to the current multiscreen configuration, then, if the `HScreenDevice` instance is currently reserved by some OCAP application other than the application that invoked the `setMultiScreenConfiguration(...)` method or reserved by any OCAP application in case this change is a result of a Host device-initiated change (without explicit invocation of this method by an OCAP application), then, for that `HScreenDevice`, and, while deferring the notification of any generated `HScreenDeviceReleasedEvent` instances, invoke the `releaseDevice()` method, and, if necessary, invoke the `release()` method of the `ResourceClient` that holds the reservation;
6. For any `HScreenDevice` instance whose `HScreenConfiguration` accessible state will or may change as a result of this change to the current multiscreen configuration, generate, but defer notification of, a corresponding `HScreenConfigurationEvent` instance;
7. For any `HScreen` instance whose `MultiScreenContext` accessible state will or may change as a result of this change to the current multiscreen configuration, generate, but defer notification of, a corresponding `MultiScreenContextEvent` instance;
8. If any `MultiScreenConfigurationListener` whose `notify()` method was invoked in step (3) above has not yet returned, then wait until either (1) all such `notify()` methods have returned or (2) a time period of at least five seconds and no greater than 30 seconds (in real-time) has elapsed since the first such method was invoked;
9. Perform all changes necessary to satisfy the postconditions specified below;

10. For each `HScreenDeviceReleasedEvent` instance generated above and in the generated order, then, for each OCAP application invoke the `statusChanged()` method of any registered `ResourceStatusListener`, providing as an argument the `HScreenDeviceReleasedEvent` instance;
11. For each `HScreenConfigurationEvent` instance generated above and in the generated order, then, for each OCAP application, invoke the `report()` method of any registered `HScreenConfigurationListener`, providing as an argument the `HScreenConfigurationEvent` instance;
12. For each `MultiScreenContextEvent` instance generated above and in the generated order, then, for each OCAP application, invoke the `notify()` method of any registered `MultiScreenContextListener`, providing as an argument the `MultiScreenContextEvent` instance;
13. Generate a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED` event for application dispatching;
14. For each OCAP application, invoke the `notify()` method of any registered `MultiScreenConfigurationListener`, providing as an argument the `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED` generated above.

7.3.4.3 Postconditions

Subsequent to a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED` event that is generated as a result of a multiscreen configuration change, the following ordered postconditions SHALL hold:

1. The Quiescent State Invariants defined below must be satisfied;
2. The current multiscreen configuration is the new configuration;
3. The new default screen must be same (Object) instance as the old default screen;
4. If default background screen devices exist in the default screen in both old and new multiscreen configurations, then, unless the application signals `allow_default_device_reconfig` as '1':
 - a. The new default background screen device must be the same instance as the old default background screen device; and
 - b. The new default background screen device must have same screen area, pixel resolution, and pixel aspect ratio as it did with the old default background screen device;

If the application signals `allow_default_device_reconfig` as '1' and a default background screen device existed in the old multiscreen configuration, but not in the new multiscreen configuration, then any reference to the old default background screen device must be reset so as to be equivalent to the empty background screen device;

5. If default video screen devices exist in the default screen in both old and new multiscreen configurations, then, unless the application signals `allow_default_device_reconfig` as '1':
 - a. The new default video screen device must be the same instance as the old default video screen device; and

- b. The new default video screen device must have same screen area, pixel resolution, and pixel aspect ratio as it did with the old default video screen device;

If the application signals `allow_default_device_reconfig` as '1' and a default video screen device existed in the old multiscreen configuration, but not in the new multiscreen configuration, then any reference to the old default video screen device must be reset so as to be equivalent to the empty video screen device;

- 6. If default graphics screen devices exist in the default screen in both old and new multiscreen configurations, then, unless the application signals `allow_default_device_reconfig` as '1':
 - a. The new default graphics screen device must be the same instance as the old default graphics screen device; and
 - b. The new default graphics screen device must have same screen area, pixel resolution, and pixel aspect ratio as it did with the old default graphics screen device;

If the application signals `allow_default_device_reconfig` as '1' and a default graphics screen device existed in the old multiscreen configuration, but not in the new multiscreen configuration, then any reference to the old default graphics screen device must be reset so as to be equivalent to the empty graphics screen device;

- 7. For every non-default screen reference obtained prior to reconfiguration, if no longer a non-default screen, then it must be equivalent to an empty screen; otherwise, it must not be equivalent to an empty screen;
- 8. For every non-default screen device reference obtained prior to reconfiguration, if no longer a non-default screen device, then it must be equivalent to an empty screen device; otherwise, it must not be equivalent to an empty screen device.

7.3.4.4 Quiescent State Invariants

Prior to a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING` event and subsequent to a corresponding `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED` event, the following ordered invariants SHALL hold:

- 1. There must be a current, non-changing multiscreen manager;
- 2. There must be a current, non-changing per-platform multiscreen configuration;
- 3. There must be a current, non-changing per-display multiscreen configuration for each display screen;
- 4. There must be a non-empty, non-changing set of accessible multiscreen configurations;
- 5. The current per-platform multiscreen configuration must be an accessible configuration (for an appropriately privileged application);
- 6. Each current per-display multiscreen configuration must be an accessible configuration (for an appropriately privileged application);
- 7. There must be a non-empty, non-changing set of screens in the current per-platform multiscreen configuration;
- 8. There must be a non-empty, non-changing set of screens in each current per-display multiscreen configuration;

9. The screens in the current per-platform multiscreen configuration must not be empty;
10. The screens in each current per-display multiscreen configuration must not be empty;
11. Any two distinct screen entries in any current multiscreen configuration must not represent the same resources;
12. There must be a current default screen;
13. The current default screen must not be equivalent to the empty screen;
14. Exactly one screen entry in some current multiscreen configuration must represent the same resources as the default screen;
15. There must be a non-empty set of accessible screens;
16. The current default screen must be a distinct member of the set of accessible screens;
17. Any background screen device of the current default screen must not be equivalent to the empty background screen device;
18. Any video screen device of the current default screen must not be equivalent to the empty video screen device;
19. Any graphics screen device of the current default screen must not be equivalent to the empty graphics screen device.

7.3.4.5 Dynamic State Invariants

Subsequent to a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING` event and prior to a corresponding `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED` event, the following ordered invariants SHALL hold:

1. There must be a current, non-changing multiscreen manager;
2. There must be a current, but possibly changing, multiscreen configuration;
3. There must be a non-empty, possibly changing, set of accessible multiscreen configurations;
4. The current multiscreen configuration must be an accessible configuration;
5. There must be a non-empty, but possibly changing, set of screens in the current multiscreen configuration;
6. The screens in the current multiscreen configuration must not be empty;
7. Any two distinct screen entries in the current multiscreen configuration must not represent the same resources;
8. There must be a current, but possibly changing, default screen;
9. The current default screen must not be equivalent to the empty screen;
10. Exactly one screen entry in the current multiscreen configuration must represent the same resources as the default screen;

11. There must be a non-empty, but possibly changing, set of accessible screens;
12. The current default screen must be a distinct member of the set of accessible screens;
13. Any background screen device of the current default screen must not be equivalent to the empty background screen device;
14. Any video screen device of the current default screen must not be equivalent to the empty video screen device;
15. Any graphics screen device of the current default screen must not be equivalent to the empty graphics screen device.

7.4 Baseline HAVi Model Extensions

The Multiple Screen Manager entails the use of certain behavioral and usage extensions to the Baseline HAVi Model, and in particular to the following HAVi defined types as described below:

- HScreen
- HScreenDevice
- HScreenConfiguration
- HScreenConfigTemplate
- HVideoDevice

7.4.1 HScreen

As specified above, each HScreen instance is characterized as a display screen or a logical screen as defined by the Multiple Screen Manager.

A distinction is made by the Multiple Screen Manager regarding an HScreen instance and the underlying resources represented by such an instance. In particular, the following underlying state associated with an HScreen instance MAY change over its temporal extent:

- The number of underlying background screen devices;
- The screen device configuration (parameter set) of these underlying background screen devices;
- The underlying background screen device that is associated with the default HBackgroundDevice instance;
- The number of underlying video screen devices;
- The screen device configuration (parameter set) of these underlying video screen devices;
- The underlying video screen device that is associated with the default HVideoDevice instance;
- The number of underlying graphics screen devices;
- The underlying graphics screen device that is associated with the default HGraphicsDevice instance;
- The screen device configuration (parameter set) of these underlying graphics screen devices.

Each HScreen instance SHALL implement the MultiScreenContext interface or, if the HScreen instance represents underlying screen resources that are configurable according to the functionality defined by MultiScreenConfigurableContext, then the HScreen instance SHALL implement the

MultiScreenConfigurableContext interface instead of MultiScreenContext (note that the former is a subinterface of the latter interface).

The HScreen.getHScreens() static method SHALL return the same value as MultiScreenManager.getInstance().getScreens().

The HScreen.getDefaultHScreen() static method SHALL return the same value as MultiScreenManager.getInstance().getDefaultScreen().

7.4.1.1 Default HScreen Assignment

When an application is initially launched, it SHALL firstly be associated with a set of accessible screens as would be returned by HScreen.getHScreens() and secondly be assigned a default HScreen according to the following rules, where the first rule that applies is used and the remainder are ignored:

1. If the ServiceContext *SC* in whose selected Service the application is signaled is associated with only one (underlying) screen, then use that screen as the default screen;
2. Otherwise, if *SC* is associated with multiple (underlying) screens, then if one of these screens is categorized as a main screen, i.e., getScreenCategory() returns MultiScreenConfiguration.SCREEN_CATEGORY_MAIN, then use the first such main screen appearing in the array of screens returned by HScreen.getHScreens();
3. Otherwise (associated with multiple screens none of which is a main screen), use the first screen appearing in the array of screens returned by HScreen.getHScreens().

7.4.1.2 Empty HScreen

MSM introduces a specially defined HScreen instance known as the *empty HScreen*, which is a unique, immutable instance of (a sub-class of) the HScreen class which is used for the purpose of establishing a well-defined, known state in a (nominally non-empty) HScreen instance.

The *empty HScreen*, referred to as *ES* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *ES* implements the following interfaces: MultiScreenContext, MultiScreenConfigurableContext, ResourceProxy, and ResourceServer;
3. *ES*.getDefaultHBackgroundDevice() SHALL return an HBackgroundDevice whose state is equivalent to the *empty HBackgroundDevice* defined below;
4. *ES*.getDefaultHVideoDevice() SHALL return an HVideoDevice whose state is equivalent to the *empty HVideoDevice* defined below;
5. *ES*.getDefaultHGraphicsDevice() SHALL return an HGraphicsDevice whose state is equivalent to the *empty HScreenDevice* defined below;
6. *ES*.getBestConfiguration(HBackgroundConfigTemplate *hbc*) SHALL return *ES*.getDefaultBackgroundDevice().getBestConfiguration(*hbc*);
7. *ES*.getBestConfiguration(HVideoConfigTemplate *hvc*) SHALL return *ES*.getDefaultVideoDevice().getBestConfiguration(*hvc*);

8. *ES*.getBestConfiguration(HGraphicsConfigTemplate *hgc*) SHALL return *ES*.getDefaultGraphicsDevice().getBestConfiguration(*hgc*);
9. *ES*.getBackgroundDevices() SHALL return new HBackgroundDevice[1] { *ES*.getDefaultHBackgroundDevice() };
10. *ES*.getVideoDevices() SHALL return new HVideoDevice[1] { *ES*.getDefaultHVideoDevice() };
11. *ES*.getGraphicsDevices() SHALL return new HGraphicsDevice[1] { *ES*.getDefaultHGraphicsDevice() };
12. *ES*.getCoherentScreenConfigurations(...) SHALL return null;
13. *ES*.setCoherentScreenConfigurations(...) SHALL throw HConfigurationException;
14. ((MultiScreenContext)*ES*).getScreenType() SHALL return MultiScreenContext.SCREEN_TYPE_LOGICAL;
15. ((MultiScreenContext)*ES*).getDisplayScreen() SHALL return null;
16. ((MultiScreenContext)*ES*).getDisplayArea() SHALL return null;
17. ((MultiScreenContext)*ES*).getOutputPorts() SHALL return null;
18. ((MultiScreenContext)*ES*).getServiceContexts() SHALL return new ServiceContext[0];
19. ((MultiScreenContext)*ES*).getVisible() SHALL return false;
20. ((MultiScreenContext)*ES*).getZOrder() SHALL return -1;
21. ((MultiScreenContext)*ES*).addScreenContextListener() SHALL NOT produce any side effect;
22. ((MultiScreenConfigurableContext)*ES*).setDisplayScreen(...) SHALL throw IllegalStateException unless SecurityException applies;
23. ((MultiScreenConfigurableContext)*ES*).setDisplayArea(...) SHALL throw IllegalStateException unless SecurityException applies;
24. ((MultiScreenConfigurableContext)*ES*).setVisible(...) SHALL throw IllegalStateException unless SecurityException applies;
25. ((MultiScreenConfigurableContext)*ES*).setZOrder(...) SHALL throw IllegalStateException unless SecurityException applies;
26. ((MultiScreenConfigurableContext)*ES*).addOutputPort(...) SHALL throw IllegalStateException unless SecurityException applies;
27. ((MultiScreenConfigurableContext)*ES*).addServiceContext(...) SHALL throw IllegalStateException unless SecurityException applies;

28. `((ResourceProxy)ES).getClient()` SHALL return null.

7.4.2 HScreenDevice

If two `HScreenDevice` instances, `SD1` and `SD2`, are equivalent with respect to their underlying resources, i.e., if `sameResources(SD1,SD2)` returns `true`, then `SD1.getIDstring()` SHALL return the same value as `SD2.getIDstring()`; otherwise, `SD1.getIDstring()` SHALL NOT return the same value as `SD2.getIDstring()`.

Any attempt by an OCAP application to reserve an `HScreenDevice` whose underlying screen device resources are not contained wholly within the set of underlying screen device resources referenced by a currently active multiscreen configuration SHALL cause an `HPermissionDeniedException` to be thrown.

Note: The above constraint is intended to prevent an application from reserving a screen device resource where that resource is not included in the set of underlying resources represented by some currently active multiscreen configuration. Such situation may apply in the case of a partition of screen device resources between an active multiscreen configuration and some non-active multiscreen configuration.

7.4.2.1 Default HScreenDevice Assignment

The default set of accessible screen devices is assigned to an application by assigning a default screen as described above. Upon default assignment to `HScreen S`, the default screen devices assigned to the application SHALL be as determined by: `S.getDefaultHBackgroundDevice()`, `S.getDefaultHVideoDevice()`, and `S.getDefaultHGraphicsDevice()`.

If the default screen has more than one screen device of a specific type, then the process for determining which of these screen devices is the default screen device of that specific type for that screen SHALL remain platform dependent, and such assignment SHALL NOT be relied upon by an interoperable application.

7.4.2.2 Empty HScreenDevice

MSM introduces a specially defined set of *empty HScreenDevice* instances of the defined sub-types: `HBackgroundDevice`, `HVideoDevice`, and `HGraphicsDevice`. These instances, referred to as *ED* below, SHALL satisfy the following constraints in addition to their sub-type specific constraints defined under the subsequent sub-headings:

1. `ED.getIDstring()` SHALL return the empty string "";
2. `ED.getScreenAspectRatio()` SHALL return `new Dimension()`;
3. `ED.addMultiScreenConfigurationListener(..)` SHALL NOT produce any side effect;
4. `ED.reserveDevice(ResourceClient)` SHALL return `false`;
5. `ED.releaseDevice()` SHALL NOT produce any side effect;
6. `ED.getClient()` SHALL return null.

7.4.2.3 Empty HBackgroundDevice

The *empty HBackgroundDevice*, referred to as *EBD* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;

2. *EBD* SHALL satisfy the constraints specified for *empty HScreenDevice* above;
3. *EBD.getDefaultConfiguration()* SHALL return the *empty HBackgroundConfiguration* as defined below;
4. *EBD.getCurrentConfiguration()* SHALL return the same value as *EBD.getDefaultConfiguration()*;
5. *EBD.getBestConfiguration(...)* SHALL return the same value as *EBD.getDefaultConfiguration()*;
6. *EBD.getConfigurations()* SHALL return new *HBackgroundConfiguration[1]* {
getDefaultConfiguration() };
7. *EBD.setBackgroundConfiguration()* SHALL throw *HConfigurationException* unless *SecurityException* or *HPermissionDeniedException* applies.

7.4.2.4 Empty HVideoDevice

The *empty HVideoDevice*, referred to as *EVD* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *EVD* SHALL satisfy the constraints specified for *empty HScreenDevice* above;
3. *EVD.getDefaultConfiguration()* SHALL return the *empty HVideoConfiguration* as defined below;
4. *EVD.getCurrentConfiguration()* SHALL return the same value as *EVD.getDefaultConfiguration()*;
5. *EVD.getBestConfiguration(...)* SHALL return the same value as *EVD.getDefaultConfiguration()*;
6. *EVD.getConfigurations()* SHALL return new *HVideoConfiguration[1]* {
getDefaultConfiguration() };
7. *EVD.setVideoConfiguration()* SHALL throw *HConfigurationException* unless *SecurityException* or *HPermissionDeniedException* applies.

7.4.2.5 Empty HGraphicsDevice

The *empty HGraphicsDevice*, referred to as *EGD* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *EGD* SHALL satisfy the constraints specified for *empty HScreenDevice* above;
3. *EGD.getDefaultConfiguration()* SHALL return the *empty HGraphicsConfiguration* as defined below;
4. *EGD.getCurrentConfiguration()* SHALL return the same value as *EGD.getDefaultConfiguration()*;

5. *EGD*.getBestConfiguration(..) SHALL return the same value as *EGD*.getDefaultConfiguration();
6. *EGD*.getConfigurations() SHALL return new HGraphicsConfiguration[1] { getDefaultConfiguration() };
7. *EGD*.setGraphicsConfiguration() SHALL throw HConfigurationException unless SecurityException or HPermissionDeniedException applies.

7.4.3 HScreenConfiguration

MSM introduces a specially defined set of *empty HScreenConfiguration* instances of the defined sub-types: HBackgroundConfiguration, HVideoConfiguration, and HGraphicsConfiguration.

7.4.3.1 Empty HScreenConfiguration

An instance of an empty HScreenConfiguration, referred to as *EC* below, SHALL satisfy the following constraints in addition to their sub-type specific constraints defined under the subsequent sub-headings:

1. *EC*.convertTo(..) SHALL return null;
2. *EC*.getFlickerFilter() SHALL return false;
3. *EC*.getInterlaced() SHALL return false;
4. *EC*.getOffset(..) SHALL return new Dimension();
5. *EC*.getPixelAspectRatio() SHALL return new Dimension();
6. *EC*.getPixelResolution() SHALL return new Dimension();
7. *EC*.getScreenArea() SHALL return new HScreenRectangle(0,0,0,0).

7.4.3.2 Empty HBackgroundConfiguration

The *empty HBackgroundConfiguration*, referred to as *EBC* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *EBC* SHALL satisfy the constraints specified for *empty HScreenConfiguration* above;
3. *EBC*.getDevice() SHALL return the *empty HBackgroundDevice* instance defined above;
4. *EBC*.getConfigTemplate() SHALL return the *empty HBackgroundConfigTemplate* defined below;
5. *EBC*.getColor() SHALL return new Color(0,0,0);
6. *EBC*.setColor() SHALL throw HConfigurationException.

7.4.3.3 Empty HVideoConfiguration

The *empty HVideoConfiguration*, referred to as *EVC* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *EVC* SHALL satisfy the constraints specified for *empty HScreenConfiguration* above;
3. *EVC*.getDevice() SHALL return the *empty HVideoDevice* instance defined above;
4. *EVC*.getConfigTemplate() SHALL return the *empty HVideoConfigTemplate* defined below.

7.4.3.4 Empty HGraphicsConfiguration

The *empty HGraphicsConfiguration*, referred to as *EGC* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *EGC* SHALL satisfy the constraints specified for *empty HScreenConfiguration* above;
3. *EGC*.getDevice() SHALL return the *empty HGraphicsDevice* instance defined above;
4. *EGC*.getConfigTemplate() SHALL return the *empty HGraphicsConfigTemplate* defined below;
5. *EGC*.getAllFonts() SHALL return new Font[0];
6. *EGC*.getCompatibleImage(Image input, HImageHints hints) SHALL return input;
7. *EGC*.getComponentHScreenRectangle(...) SHALL return null;
8. *EGC*.getPixelCoordinatesHScreenRectangle(...) SHALL return new Rectangle();
9. *EGC*.getPunchThroughToBackgroundColor(...) SHALL return null;
10. *EGC*.dispose(Color) SHALL NOT produce any side effect.

7.4.4 HScreenConfigTemplate

MSM introduces a specially defined set of *empty HScreenConfigTemplate* instances of the defined sub-types: HBackgroundConfigTemplate, HVideoConfigTemplate, and HGraphicsConfigTemplate.

7.4.4.1 Empty HScreenConfigTemplate

An instance of an empty HScreenConfigTemplate, referred to as *ET* below, SHALL satisfy the following constraints in addition to their sub-type specific constraints defined under the subsequent sub-headings:

1. *ET*.getPreferenceObject(int) SHALL throw IllegalArgumentException;
2. *ET*.getPreferencePriority(int) SHALL return HScreenConfigTemplate.DONT_CARE;
3. *ET*.setPreference(int, int) SHALL throw an IllegalArgumentException;
4. *ET*.setPreference(int, Object, int) SHALL throw an IllegalArgumentException.

7.4.4.2 *Empty HBackgroundConfigTemplate*

The *empty HBackgroundConfigTemplate*, referred to as *EBT* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *EBT* SHALL satisfy the constraints specified for *empty HScreenConfigTemplate* above;
3. *EBT.isConfigSupported(HBackgroundConfiguration)* SHALL return false.

7.4.4.3 *Empty HVideoConfigTemplate*

The *empty HVideoConfigTemplate*, referred to as *EVT* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *EVT* SHALL satisfy the constraints specified for *empty HScreenConfigTemplate* above;
3. *EVT.isConfigSupported(HVideoConfiguration)* SHALL return false.

7.4.4.4 *Empty HGraphicsConfigTemplate*

The *empty HGraphicsConfigTemplate*, referred to as *EGT* below, SHALL satisfy the following constraints:

1. Within the scope of an application's reference, it is a unique (object) instance;
2. *EGT* SHALL satisfy the constraints specified for *empty HScreenConfigTemplate* above;
3. *EGT.isConfigSupported(HGraphicsConfiguration)* SHALL return false.

7.4.5 **HVideoDevice**

If the use of some mechanism defined by this specification permits more than one video pipeline (consisting of at least a video decoder and a decoder format conversion component) to be associated with (mapped to) the underlying video screen device resources represented by an *HVideoDevice* instance, then one of these pipelines is designated as the contributing video pipeline and the remainder are designated as non-contributing video pipelines. A non-contributing video pipeline SHALL NOT contribute video information or related audio information to the underlying video screen device resources represented by any *HVideoDevice* instance.

The contributing video pipeline SHALL be determined from a group of potentially contributing video pipelines according to the following ordered rules:

1. If only one video pipeline among the candidate video pipelines is associated with the active video elementary stream of some non-abstract service, then that video pipeline is the contributing pipeline;
2. otherwise, if more than one video pipeline among the candidate video pipelines is associated with the active video elementary stream of some non-abstract service, then the video pipeline of the non-abstract service most recently selected is the contributing pipeline;
3. otherwise, if only one video pipeline among the candidate video pipelines is associated with a media player controlled by some non-abstract service, then the video pipeline of that media player is the contributing pipeline;

4. otherwise, if more than one video pipeline among the candidate video pipelines is associated with a media player of some non-abstract service, then the video pipeline of the media player most recently started (or resumed after being paused or stopped) is the contributing pipeline;
5. otherwise, if only one video pipeline among the candidate video pipelines is associated with a media player controlled by some abstract service, then the video pipeline of that media player is the contributing pipeline;
6. otherwise, if more than one video pipeline among the candidate video pipelines is associated with a media player of some abstract service, then the video pipeline of the media player most recently started (or resumed after being paused or stopped) is the contributing pipeline;
7. otherwise, the determination of which video pipeline is the contributing pipeline is not defined by this specification, and is implementation dependent.

7.5 Security

The invocation of certain methods defined by this specification requires the application to have either (or both) `MonitorAppPermission("multiscreen.configuration")` or (and) `MonitorAppPermission("multiscreen.context")`. In OCAP Host devices that implement the OCAP MSM Extension, the following additional permissions are considered to apply to `MonitorAppPermission`:

Table 7-1 - Additional Monitor Application Permissions for Multiscreen Manager Extension

Permission Name	What the Permission Allows	Description
<code>multiscreen.configuration</code>	Provides ability to access and modify platform multiscreen configuration state.	Applications with this permission may modify the platform's multiscreen configuration and discover and monitor its dynamic state.
<code>multiscreen.context</code>	Provides ability to modify a screen's multiscreen context state.	Applications with this permission may modify the multiscreen context state of accessible screens.

In addition, the enumerated token value type of the `name` attribute of the `ocap:monitorapplication` element type defined by the Document Type Definition (DTD) of the Permission Request File (PRF) SHALL be considered to contain the following values: `multiscreen.configuration` and `multiscreen.context`.

7.6 System Property

If an OCAP Host device implements the MSM Extension, then the `"ocap.api.option.msm"` Java system property SHALL be defined with a value corresponding to the implemented version of this specification, where this version of this specification is `"1.0.0"`. Conversely, if a Host device defines the `"ocap.api.option.msm"` Java system property, then the Host device SHALL implement the version of this specification that corresponds with the property's value.

Read access to this property SHALL be granted to both signed and unsigned applications.

7.7 Registry of Constants

This subsection contains the registry of Java constants specific to this extension.

org.ocap.*

org.ocap.ui.MultiScreenConfigurableContext		
public static final int	CONFIGURABLE_SCREEN_PARAMETER_AUDIO_FOCUS	1
public static final int	CONFIGURABLE_SCREEN_PARAMETER_AUDIO_SOURCE	0
public static final int	CONFIGURABLE_SCREEN_PARAMETER_DEVICE_Z_ORDER	2
public static final int	CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_AREA	3
public static final int	CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_SCREEN	4
public static final int	CONFIGURABLE_SCREEN_PARAMETER_OUTPUT_PORT	5
public static final int	CONFIGURABLE_SCREEN_PARAMETER_SERVICE_CONTEXT	6
public static final int	CONFIGURABLE_SCREEN_PARAMETER_VISIBILITY	7
public static final int	CONFIGURABLE_SCREEN_PARAMETER_Z_ORDER	8

org.ocap.ui.MultiScreenConfiguration		
public static final java.lang.String	SCREEN_CATEGORY_DISPLAY	"display"
public static final java.lang.String	SCREEN_CATEGORY_GENERAL	"general"
public static final java.lang.String	SCREEN_CATEGORY_MAIN	"main"
public static final java.lang.String	SCREEN_CATEGORY_NONE	"none"
public static final java.lang.String	SCREEN_CATEGORY_OVERLAY	"overlay"
public static final java.lang.String	SCREEN_CATEGORY_PIP	"pip"
public static final java.lang.String	SCREEN_CATEGORY_POP	"pop"
public static final java.lang.String	SCREEN_CONFIGURATION_DISPLAY	"display"
public static final java.lang.String	SCREEN_CONFIGURATION_GENERAL	"general"
public static final java.lang.String	SCREEN_CONFIGURATION_NON_PIP	"non-pip"
public static final java.lang.String	SCREEN_CONFIGURATION_PIP	"pip"
public static final java.lang.String	SCREEN_CONFIGURATION_POP	"pop"

org.ocap.ui.MultiScreenContext		
public static final int	SCREEN_TYPE_DISPLAY	0
public static final int	SCREEN_TYPE_LOGICAL	1

org.ocap.ui.event.MultiScreenConfigurationEvent		
public static final int	MULTI_SCREEN_CONFIGURATION_CHANGED	1
public static final int	MULTI_SCREEN_CONFIGURATION_CHANGING	0
public static final int	MULTI_SCREEN_CONFIGURATION_LAST	3

org.ocap.ui.event.MultiScreenConfigurationEvent		
public static final int	MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED	2
public static final int	MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED	3

org.ocap.ui.event.MultiScreenContextEvent		
public static final int	MULTI_SCREEN_CONTEXT_AUDIO_FOCUS_CHANGED	41
public static final int	MULTI_SCREEN_CONTEXT_AUDIO_SOURCES_CHANGED	40
public static final int	MULTI_SCREEN_CONTEXT_DEVICES_CHANGED	32
public static final int	MULTI_SCREEN_CONTEXT_DEVICES_Z_ORDER_CHANGED	33
public static final int	MULTI_SCREEN_CONTEXT_DISPLAY_AREA_CHANGED	36
public static final int	MULTI_SCREEN_CONTEXT_DISPLAY_SCREEN_CHANGED	35
public static final int	MULTI_SCREEN_CONTEXT_OUTPUT_PORT_CHANGED	37
public static final int	MULTI_SCREEN_CONTEXT_SERVICE_CONTEXT_CHANGED	34
public static final int	MULTI_SCREEN_CONTEXT_VISIBILITY_CHANGED	38
public static final int	MULTI_SCREEN_CONTEXT_Z_ORDER_CHANGED	39
public static final int	MULTI_SCREEN_CONTEXTS_LAST	41

org.ocap.ui.event.MultiScreenEvent		
public static final int	MULTI_SCREEN_CONFIGURATION_FIRST	0
public static final int	MULTI_SCREEN_CONTEXT_FIRST	32

org.ocap.ui.event.MultiScreenResourceEvent		
public static final int	MULTI_SCREEN_RESOURCE_SCREEN_RELEASED	0
public static final int	MULTI_SCREEN_RESOURCE_SCREEN_RESERVED	0

Annex A Multiscreen Manager API

Package org.ocap.ui

Extensions to HAVi User Interface functionality.

Interface Summary

MultiScreenConfigurableContext	This interface provides a set of tools for accomplishing the following:
MultiScreenConfiguration	The <code>MultiScreenConfiguration</code> interface, implemented by an OCAP host platform, provides information on a discrete screen configuration as well as a mechanism for monitoring changes to the screen configuration.
MultiScreenContext	This interface provides a set of tools for accomplishing the following:

Class Summary

MultiScreenManager	The <code>MultiScreenManager</code> class is an abstract, singleton management class implemented by an OCAP host platform that provides multiscreen management services.
---------------------------	--

org.ocap.ui**Interface MultiScreenConfigurableContext****All Superinterfaces:**

MultiScreenContext, org.davic.resources.ResourceProxy

```
public interface MultiScreenConfigurableContext
extends MultiScreenContext, org.davic.resources.ResourceProxy
```

This interface provides a set of tools for accomplishing the following:

1. modifying the mapping of logical HScreens to display HScreens including the area (extent) on the display HScreen where a logical HScreen appears, its visibility, and its z-order (among other HScreens);
2. modifying the z-order of HScreenDevices within an HScreen;
3. modifying the set of ServiceContexts associated with an HScreen;
4. modifying the association of display HScreens and corresponding VideoOutputPort instances;
5. modifying the set of HScreenDevices whose generated audio constitute the set of audio sources of an HScreen;
6. modifying the current audio focus assignment of a display HScreen;
7. reserving and releasing reservation of underlying screen and screen device resources;
8. obtaining a reference to current resource client that has reserved screen and its underlying resources;
9. establishing the currently active per-display multiscreen configuration of a display HScreen;

If an HScreen instance may be exposed to an OCAP application and if that HScreen is configurable with respect to the functionality defined by this interface, then an MSM implementation SHALL support the MultiScreenConfigurableContext interface on every such HScreen instance.

An MSM implementation MAY support this interface on an HScreen instance that is not configurable with respect to the functionality defined by this interface.

A given implementation of this interface is not required to support any or all defined configuration changes, but MAY, due to hardware or other constraints, support only specific configuration changes. If an implementation of this interface does not support a specific configuration change, then an attempt to perform that change SHALL cause an IllegalStateException to be raised, as described under each method defined below.

Since:

MSM I01

Field Summary	
static int	CONFIGURABLE_SCREEN_PARAMETER_AUDIO_FOCUS Configurable parameter identifying configurability of audio focus.
static int	CONFIGURABLE_SCREEN_PARAMETER_AUDIO_SOURCE Configurable parameter identifying configurability of audio source(s).

Field Summary	
static int	CONFIGURABLE_SCREEN_PARAMETER_DEVICE_Z_ORDER Configurable parameter identifying configurability of screen device z-order.
static int	CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_AREA Configurable parameter identifying configurability of screen's associated display area.
static int	CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_SCREEN Configurable parameter identifying configurability of screen's associated display screen.
static int	CONFIGURABLE_SCREEN_PARAMETER_OUTPUT_PORT Configurable parameter identifying configurability of screen's associated output port(s).
static int	CONFIGURABLE_SCREEN_PARAMETER_SERVICE_CONTEXT Configurable parameter identifying configurability of screen's associated service context(s).
static int	CONFIGURABLE_SCREEN_PARAMETER_VISIBILITY Configurable parameter identifying configurability of screen's visibility.
static int	CONFIGURABLE_SCREEN_PARAMETER_Z_ORDER Configurable parameter identifying configurability of screen's z-order.

Fields inherited from interface org.ocap.ui.MultiScreenContext
SCREEN_TYPE_DISPLAY, SCREEN_TYPE_LOGICAL

Method Summary	
void	addAudioSources (org.havi.ui.HScreenDevice[] devices, boolean mixWithAudioFocus) Add audio source(s) for this screen.
void	addOutputPorts (org.ocap.hardware.VideoOutputPort[] ports, boolean removeExisting) Add video output port(s) to which screen is mapped.
void	addServiceContexts (javax.tv.service.selection.ServiceContext[] contexts, boolean associateDefaultDevices) Add service context(s) to this screen.
void	assignAudioFocus () Assign audio focus to this screen.
boolean	checkServiceContextCompatibility (javax.tv.service.selection.ServiceContext context) Test compatibility of service context with screen.
org.davic.resources.ResourceClient	getClient () Obtain the resource client that currently holds the reservation on the underlying screen and screen resources associated with this HScreen instance.
java.lang.Object[]	getDiscreteParameterSpace (int parameter) Obtain the discrete, (sub)set of values of the value type space of a configurable parameter.
boolean	hasDiscreteParameterSpace (int parameter) Determine if a supported configurable parameter has a discrete or continuously variable value space.

Method Summary

boolean	isConfigurableParameter (int parameter) Determine if configurable parameter is supported as configurable (as opposed to fixed) by the platform implementation for some screen.
void	releaseScreen () Atomically release underlying resources of screen.
void	removeAudioSources (org.havi.ui.HScreenDevice[] devices) Remove audio source(s) for this screen.
void	removeOutputPorts (org.ocap.hardware.VideoOutputPort[] ports) Remove video output port(s) to which screen is mapped.
void	removeServiceContexts (javax.tv.service.selection.ServiceContext[] contexts) Remove service context(s) from screen.
void	requestMultiScreenConfigurationChange (MultiScreenConfiguration configuration, java.util.Dictionary serviceContextAssociations) Request that the currently active multiscreen configuration for this display screen be changed.
boolean	reserveScreen (org.davic.resources.ResourceClient client, java.lang.Object requestData) Atomically reserve underlying resources of this screen.
void	setDisplayArea (org.havi.ui.HScreenRectangle rect) Set area of display screen to which logical screen is mapped.
void	setDisplayScreen (org.havi.ui.HScreen screen) Associate logical screen with display screen.
void	setMultiScreenConfiguration (MultiScreenConfiguration configuration, java.util.Dictionary serviceContextAssociations) Set currently active multiscreen configuration for this display screen (i.e., choose among the set of subsets of logical screens that may be associated with this display screen at a given time).
void	setVisible (boolean visible) Set screen visibility.
void	setZOrder (org.havi.ui.HScreenDevice[] devices) Set the screen device z-order within this screen for a set of screen devices.
void	setZOrder (int order) Set screen z-order.

Methods inherited from interface org.ocap.ui.MultiScreenContext

addMultiScreenConfigurationListener, addScreenContextListener, getAudioFocus, getAudioSources, getDisplayArea, getDisplayScreen, getID, getMultiScreenConfiguration, getMultiScreenConfigurations, getMultiScreenConfigurations, getOutputPorts, getScreenCategory, getScreenType, getServiceContexts, getVisible, getZOrder, getZOrder, removeMultiScreenConfigurationListener, removeScreenContextListener

Field Detail

CONFIGURABLE_SCREEN_PARAMETER_AUDIO_SOURCE

`static final int CONFIGURABLE_SCREEN_PARAMETER_AUDIO_SOURCE`

Configurable parameter identifying configurability of audio source(s).

Configuration of the audio source(s) of a screen is accomplished by using the `addAudioSources(...)` and `removeAudioSources(...)` methods defined by this interface.

If the `HScreen` instance referenced by this interface supports configuration of its audio source(s), then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_AUDIO_SOURCE)` SHALL return `true`.

If `hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_AUDIO_SOURCE)` returns `true`, then `getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_AUDIO_SOURCE)` SHALL return a value of type `HScreenDevice[]`, where each entry in the value array is an accessible screen device of this screen that can serve as an audio source for this screen.

Since:
MSM I01

CONFIGURABLE_SCREEN_PARAMETER_AUDIO_FOCUS

`static final int CONFIGURABLE_SCREEN_PARAMETER_AUDIO_FOCUS`

Configurable parameter identifying configurability of audio focus.

Configuration of the audio focus of a screen is accomplished by using the `assignAudioFocus(...)` method defined by this interface.

If the `HScreen` instance referenced by this interface supports configuration of its audio source(s), then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_AUDIO_FOCUS)` SHALL return `true`.

If `hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_AUDIO_FOCUS)` returns `true`, then `getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_AUDIO_FOCUS)` SHALL return a value of type `HScreen[]`, where each entry in the value array is an accessible screen that can serve as an audio focus screen.

If this screen is a display screen, then the returned entries SHALL be restricted to those logical screens that are currently mapped to this display screen that can be assigned audio focus. Because a display screen can always be assigned audio focus directly (as opposed to assigning audio focus to some logical screen mapped to the display screen), the display screen is not itself included in the returned entries.

If this screen is a logical screen and if this logical screen is mapped to a display screen and is capable of being assigned audio focus, then the returned array SHALL contain only this screen; otherwise, the returned array SHALL be empty.

Since:
MSM I01

CONFIGURABLE_SCREEN_PARAMETER_DEVICE_Z_ORDER

`static final int CONFIGURABLE_SCREEN_PARAMETER_DEVICE_Z_ORDER`

Configurable parameter identifying configurability of screen device z-order.

Configuration of device z-order of a screen is accomplished by using the `setZOrder(HScreenDevice[])` method defined by this interface.

If the `HScreen` instance referenced by this interface supports configuration of the z-order of its screen device(s), then

`isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_DEVICE_Z_ORDER)` SHALL return true.

If

`hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_DEVICE_Z_ORDER)` returns true, then

`getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_DEVICE_Z_ORDER)`

SHALL return a value of type `HScreenDevice[][]`, where each entry in the value array is an array of screen devices whose order matches a supported z-ordering of those screen devices, where the first entry of such an array of screen devices is back-most in z-order (i.e., device z-order of zero).

Example:

The following assumptions apply:

1. screen has one background device (B1), one video device (V1), and two graphics devices (G1 and G2);
2. two orderings of graphics devices are supported: (1) B1<V1<G1<G2 and (2) B1<V1<G2<G1;

```
MultiScreenConfigurableContext msxx = (MultiScreenConfigurableContext)
HScreen.getDefaultHScreen();
Object[] values =
msxx.getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_DEVICE_Z_OR
DER);
ASSERT ( values != null );
ASSERT ( values instanceof HScreenDevice[][] );
ASSERT ( values.length == 2 );
HScreenDevice[] order1 = (HScreenDevice[]) values[0];
ASSERT ( order1 != null );
ASSERT ( order1.length == 4 );
ASSERT ( order1[0] instanceof HBackgroundDevice );
ASSERT ( order1[1] instanceof HVideoDevice );
ASSERT ( order1[2] instanceof HGraphicsDevice );
ASSERT ( order1[2].getIDstring().equals ( "G1" ) );
ASSERT ( order1[3] instanceof HGraphicsDevice );
ASSERT ( order1[3].getIDstring().equals ( "G2" ) );
HScreenDevice[] order2 = (HScreenDevice[]) values[1];
ASSERT ( order2 != null );
ASSERT ( order2.length == 4 );
ASSERT ( order2[0] instanceof HBackgroundDevice );
ASSERT ( order2[1] instanceof HVideoDevice );
ASSERT ( order2[2] instanceof HGraphicsDevice );
ASSERT ( order2[2].getIDstring().equals ( "G2" ) );
ASSERT ( order2[3] instanceof HGraphicsDevice );
ASSERT ( order2[3].getIDstring().equals ( "G1" ) );
```

Since:

MSM I01

CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_AREA

```
static final int CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_AREA
```

Configurable parameter identifying configurability of screen's associated display area.

Configuration of the display area of a (logical) screen is accomplished by using the `setDisplayArea(HScreenRectangle)` method defined by this interface.

If the `HScreen` instance referenced by this interface is a display screen, then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_AREA)` SHALL return `false`; otherwise, if this logical screen supports configuration of its display area, then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_AREA)` SHALL return `true`.

If `hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_AREA)` returns `true`, then `getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_AREA)` SHALL return a value of type `HScreenRectangle[]`, where each entry in the value array is a supported display area.

Since:
MSM I01

CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_SCREEN

`static final int CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_SCREEN`

Configurable parameter identifying configurability of screen's associated display screen.

Configuration of the display screen of a (logical) screen is accomplished by using the `setDisplayScreen(HScreen)` method defined by this interface.

If the `HScreen` instance referenced by this interface is a display screen, then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_SCREEN)` SHALL return `false`; otherwise, if this logical screen supports configuration of its display screen, then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_SCREEN)` SHALL return `true`.

If `hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_SCREEN)` returns `true`, then `getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_DISPLAY_SCREEN)` SHALL return a value of type `HScreen[]`, where each entry in the value array is an accessible display screen to which this logical screen MAY be mapped.

Since:
MSM I01

CONFIGURABLE_SCREEN_PARAMETER_OUTPUT_PORT

`static final int CONFIGURABLE_SCREEN_PARAMETER_OUTPUT_PORT`

Configurable parameter identifying configurability of screen's associated output port(s).

Configuration of the output port(s) of a screen is accomplished by using the `addOutputPorts(...)` and `removeOutputPorts(...)` methods defined by this interface.

If the `HScreen` instance referenced by this interface supports configuration of its video output port(s), then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_OUTPUT_PORT)` SHALL return `true`.

If `hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_OUTPUT_PORT)` returns `true`, then

`getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_OUTPUT_PORT)`
SHALL return a value of type `VideoOutputPort []`, where each entry in the value array is an accessible video output port to which this screen may be directly mapped.

Since:
MSM I01

CONFIGURABLE_SCREEN_PARAMETER_SERVICE_CONTEXT

`static final int CONFIGURABLE_SCREEN_PARAMETER_SERVICE_CONTEXT`

Configurable parameter identifying configurability of screen's associated service context(s).

Configuration of the service context(s) of a screen is accomplished by using the `addServiceContexts(. .)` and `removeServiceContexts(. .)` methods defined by this interface, and by using the `swapServiceContexts(. .)` and `moveServiceContexts(. .)` methods defined by `MultiScreenManager`.

If the `HScreen` instance referenced by this interface supports configuration of its output port(s), then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_SERVICE_CONTEXT)` SHALL return `true`.

If `hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_SERVICE_CONTEXT)` returns `true`, then `getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_SERVICE_CONTEXT)` SHALL return a value of type `ServiceContext []`, where each entry in the value array is an accessible service context that can be associated with this screen.

Since:
MSM I01

CONFIGURABLE_SCREEN_PARAMETER_VISIBILITY

`static final int CONFIGURABLE_SCREEN_PARAMETER_VISIBILITY`

Configurable parameter identifying configurability of screen's visibility.

Configuration of the visibility of a screen is accomplished by using the `setVisible(boolean)` method defined by this interface.

If the `HScreen` instance referenced by this interface supports configuration of its visibility, then `isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_VISIBILITY)` SHALL return `true`, but `hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_VISIBILITY)` SHALL return `false`, implying that if visibility is configurable, then a continuous parameter space (i.e., both `true` and `false`) applies.

Since:
MSM I01

CONFIGURABLE_SCREEN_PARAMETER_Z_ORDER

`static final int CONFIGURABLE_SCREEN_PARAMETER_Z_ORDER`

Configurable parameter identifying configurability of screen's z-order.

Configuration of the z-order of a screen is accomplished by using the `setZOrder(int)` method defined by this interface.

If the `HScreen` instance referenced by this interface supports configuration of its z-order (with respect to other screens within its multiscreen configuration), then

`isConfigurableParameter(CONFIGURABLE_SCREEN_PARAMETER_Z_ORDER)` SHALL return true.

If `hasDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_Z_ORDER)` returns true, then `getDiscreteParameterSpace(CONFIGURABLE_SCREEN_PARAMETER_Z_ORDER)` SHALL return a value of type `Integer[]`, where each value entry v in the value array is such that `v.intValue()` returns a supported z-order index for this screen in the context of this screen's multiscreen configuration.

Since:
MSM I01

Method Detail

isConfigurableParameter

boolean **isConfigurableParameter**(int parameter)

Determine if configurable parameter is supported as configurable (as opposed to fixed) by the platform implementation for some screen.

Parameters:

parameter - a configurable screen parameter enumeration value as defined above.

Returns:

If the platform implementation supports either continuous or discrete variation of the specified *parameter* on this screen, then returns true; otherwise, returns false.

Since:
MSM I01

hasDiscreteParameterSpace

boolean **hasDiscreteParameterSpace**(int parameter)
throws `java.lang.IllegalArgumentException`

Determine if a supported configurable parameter has a discrete or continuously variable value space.

In the present context, a "continuously" configurable parameter means the platform supports or can approximate all values of the parameter's value type, while "discrete" means only certain, enumerable values of the parameter's value type may be used as reported by `getDiscreteParameterSpace(...)`.

Parameters:

parameter - a configurable screen parameter enumeration value as defined above.

Returns:

If the platform implementation supports a discrete, (sub)set of values of the value type space of the specified *parameter* on this screen, then returns true; otherwise, returns false, in which case all values of the value type space are supported (or approximated).

Throws:

`java.lang.IllegalArgumentException` - if `isConfigurableParameter(parameter)` returns false.

Since:
MSM I01

getDiscreteParameterSpace

`java.lang.Object[]` **getDiscreteParameterSpace**(int parameter)
throws

`java.lang.IllegalArgumentException`

Obtain the discrete, (sub)set of values of the value type space of a configurable parameter.

The actual runtime type of the array and the array's elements returned by this method SHALL be as defined for the specified configurable parameter as documented by each configurable parameter's specification above.

Unless indicated otherwise by the definition of a specific configurable parameter, the order of entries in the array returned by this method is not defined by this specification, and SHALL be considered implementation dependent by an interoperable application.

A set of supported discrete parameters MAY change for a given screen and a given configurable parameter over the lifetime of a screen based upon the dynamic state of the screen's underlying resources; however, such a change, if it occurs, SHALL NOT occur outside the time interval between the completion of dispatching

`MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING` event and the completion of dispatching the corresponding

`MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED` event.

Parameters:

`parameter` - a configurable screen parameter enumeration value as defined above.

Returns:

an array of object instances, each of which denote a discrete value of the specified parameter that is supported (or approximatable) by the platform.

Throws:

`java.lang.IllegalArgumentException` - if `isConfigurableParameter(parameter)` or `hasDiscreteParameterSpace(parameter)` returns false.

Since:

MSM I01

setVisible

```
void setVisible(boolean visible)
    throws java.lang.SecurityException,
           java.lang.IllegalStateException
```

Set screen visibility.

If this screen is a logical screen, then cause it to be marked as visible (if previously hidden) or hidden (if previously visible); otherwise, if this screen is a display screen, then cause all logical screens mapped to this display screen to be marked as visible (if previously hidden) or hidden (if previously visible).

Parameters:

`visible` - a boolean value indicating whether this logical `HScreen` or the logical screens mapped to this display `HScreen` should be made visible or hidden (non-visible) on its associated display `HScreen`.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalStateException` - if the visibility for this `HScreen` cannot be changed, e.g., if the platform uses a permanent visibility setting.

Since:

MSM I01

setZOrder

```
void setZOrder(int order)
    throws java.lang.SecurityException,
           java.lang.IllegalStateException
```

Set screen z-order.

Cause this logical HScreen to change its z-order among other logical HScreens mapped to the same display HScreen.

Parameters:

order - a positive integer value indicating the new z-order to assign to this logical HScreen.

Throws:

java.lang.SecurityException - if the calling thread has not been granted

MonitorAppPermission("multiscreen.context").

java.lang.IllegalStateException - if this HScreen's type is not SCREEN_TYPE_LOGICAL or if the z-order for this HScreen cannot be changed, e.g., if the platform uses a permanent z-order setting.

Since:

MSM I01

setZOrder

```
void setZOrder(org.havi.ui.HScreenDevice[] devices)
    throws java.lang.SecurityException,
           java.lang.IllegalArgumentException,
           java.lang.IllegalStateException
```

Set the screen device z-order within this screen for a set of screen devices.

Atomically set the z-order of the specified set of HScreenDevice within this screen where the following constraints apply:

- if an HBackgroundDevice is present in the specified set of devices, then it (1) precedes any HVideoDevice contained in the set of devices and (2) precedes any HGraphicsDevice contained in the set of devices;
- if an HVideoDevice is present in the specified set of devices, then it precedes any HGraphicsDevice contained in the set of devices;

If no exception is thrown by this method, then the set of specified HScreenDevice instances will be ordered such that MultiScreenContext.getZOrder(HScreenDevice) when invoked on this screen with any of the specified devices will return a z-order index that preserves the relative order of the specified devices.

If fewer than the entire set of HScreenDevice instances associated with this screen is provided in the *devices* argument, then the resulting relative order of unspecified devices with respect to specified devices is not defined, except that the constraints defined by MultiScreenContext.getZOrder(HScreenDevice) SHALL apply to the total ordering of devices in this screen after this method returns.

Parameters:

devices - an ordered array of HScreenDevice instances that are associated with this screen.

Throws:

java.lang.SecurityException - if the calling thread has not been granted

MonitorAppPermission("multiscreen.context").

java.lang.IllegalArgumentException - if *device* is not an HScreenDevice of this screen.

java.lang.IllegalStateException - if (1) the z-order for the specified HScreenDevice cannot be changed, e.g., if the platform uses a permanent z-order setting for screen devices in this screen, or (2) the order of specified devices does not permit the assignment of z-order indices that satisfies the above constraints.

Since:

MSM I01

addAudioSources

```
void addAudioSources(org.havi.ui.HScreenDevice[] devices,  
                    boolean mixWithAudioFocus)  
    throws java.lang.SecurityException,  
           java.lang.IllegalArgumentException,  
           java.lang.IllegalStateException
```

Add audio source(s) for this screen.

Add one or more `HScreenDevice` instances to the set of audio sources from which presented audio is selected (and mixed) for the purpose of audio presentation from this screen.

If a specified audio source is already designated as an audio source for this screen, but *mixWithAudioFocus* differs from that specified when it was added as an audio source, then the new *mixWithAudioFocus* value applies.

Parameters:

`devices` - a non-empty array of `HScreenDevice` instances, where each such instance contributes to a mixed, audio presentation from this screen.

`mixWithAudioFocus` - if `true`, then the specified screen devices contribute (mix) audio to (with) any audio output associated with any video output port with which this screen is associated (directly or indirectly) regardless of whether or not this screen is assigned audio focus; if `false`, then they contribute audio only when this screen is assigned audio focus.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalArgumentException` - if `devices` is not a non-empty array or some entry of `devices` is not an `HScreenDevice` of this screen.

`java.lang.IllegalStateException` - if (1) the audio sources for this `HScreen` cannot be changed, e.g., if the platform uses a permanent audio source setting for screen devices in this screen, or (2) if multiple audio sources are specified and audio mixing is not supported.

Since:

MSM I01

removeAudioSources

```
void removeAudioSources(org.havi.ui.HScreenDevice[] devices)  
    throws java.lang.SecurityException,  
           java.lang.IllegalArgumentException,  
           java.lang.IllegalStateException
```

Remove audio source(s) for this screen.

Removes all or some non-empty set of specific `HScreenDevice` instances from the set of audio sources of this `HScreen`. If `devices` is `null`, then all audio sources are removed.

Parameters:

`devices` - either `null` or a non-empty set of `HScreenDevice` instances.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalArgumentException` - if `devices` is not `null` and some `HScreenDevice` entry is not associated with this `HScreen` instance.

`java.lang.IllegalStateException` - if a specified `HScreenDevice` used as an audio source for this `HScreen` cannot be changed, e.g., if the platform uses a permanent association of audio sources with the specified `HScreenDevice`.

Since:

MSM I01

assignAudioFocus

```
void assignAudioFocus( )
    throws java.lang.SecurityException,
           java.lang.IllegalStateException
```

Assign audio focus to this screen.

At any given time, a display screen **SHALL** assign audio focus to itself or exactly one logical screen that maps to it (the display screen). When audio focus is (newly) assigned to a logical screen of a display screen and that logical screen does not currently have audio focus assigned to it, then audio focus **SHALL** be removed from any other logical screen that is mapped to that display screen and assigned instead to the newly assigned logical screen.

If no logical screen is mapped to a given display screen or no logical screen mapped to a given display screen is assigned audio focus, then the display screen **SHALL** assign itself audio focus (by default). Audio focus **MAY** be explicitly assigned to a display screen by using this method on a display screen instance.

The audio focus screen of a display screen is the screen whose currently selected audio sources are assigned to be rendered on all (implied) audio presentation devices of all video output ports to which the display screen is mapped. If the screen to which audio focus is assigned has no audio source, i.e., has an empty set of audio sources, then audio (as produced by or potentially produced by the OCAP platform) **SHALL NOT** be rendered by any (implied) audio presentation device of all video output ports to which the display screen is mapped.

Note: Each distinct display screen may have a distinct logical screen to which audio focus is assigned for the purpose of rendering audio of the display screen (and its collection of mapped logical screens).

Throws:

java.lang.SecurityException - if the calling thread has not been granted MonitorAppPermission("multiscreen.context").

java.lang.IllegalStateException - if audio focus cannot be assigned to this HScreen or the current audio focus cannot be changed.

Since:

MSM I01

addOutputPorts

```
void addOutputPorts(org.ocap.hardware.VideoOutputPort[] ports,
                    boolean removeExisting)
    throws java.lang.SecurityException,
           java.lang.IllegalStateException
```

Add video output port(s) to which screen is mapped.

If this HScreen is a logical screen rather than a display screen, then it **SHALL** be considered to function as equivalent to a display screen for the purpose of mapping to the specified video output port(s); i.e., the logical screen is treated as if it were a main display screen on its own accord.

Parameters:

ports - a non-empty array of VideoOutputPort instances.

removeExisting - if true, then remove association with existing screen (if such association exists) before adding to new screen;

Throws:

java.lang.SecurityException - if the calling thread has not been granted MonitorAppPermission("multiscreen.context").

java.lang.IllegalStateException - if (1) the specified VideoOutputPort is already associated with a different HScreen and *removeExisting* is not true, (2) this HScreen cannot be

mapped to some specified `VideoOutputPort` due to some platform specific hardware constraint, or (3) the set of `VideoOutputPort` instances to which this `HScreen` is mapped cannot be changed, e.g., if the platform uses a permanent association with a specific set of `VideoOutputPort` instances.

Since:

MSM I01

removeOutputPorts

```
void removeOutputPorts(org.ocap.hardware.VideoOutputPort[] ports)
    throws java.lang.SecurityException,
           java.lang.IllegalArgumentException,
           java.lang.IllegalStateException
```

Remove video output port(s) to which screen is mapped.

Removes all or some non-empty set of specific `VideoOutputPort` instances from the set of video output ports to which this `HScreen` is mapped. If `ports` is null, then all video output ports associations are removed.

Parameters:

`ports` - either null or a non-empty array of `VideoOutputPort` instances.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalArgumentException` - if `ports` is not null and some `VideoOutputPort` entry is not associated with this `HScreen` instance.

`java.lang.IllegalStateException` - if a specified `VideoOutputPort` for this `HScreen` cannot be changed, e.g., if the platform uses a permanent association with the specified `VideoOutputPort`.

Since:

MSM I01

setDisplayScreen

```
void setDisplayScreen(org.havi.ui.HScreen screen)
    throws java.lang.SecurityException,
           java.lang.IllegalStateException
```

Associate logical screen with display screen.

Associates this logical `HScreen` with a display `HScreen`.

Parameters:

`screen` - a display `HScreen` instance or null. If null, and if this method executes without an exception, then upon return, this logical `HScreen` is no longer associated with a display `HScreen`.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalStateException` - if (1) this `HScreen`'s type is not `SCREEN_TYPE_LOGICAL`, (2) the specified `screen` is not null and its screen type is not `SCREEN_TYPE_DISPLAY`, or (3) the display `HScreen` associated with this logical `HScreen` cannot be changed, e.g., if the platform uses a permanent association with a specific display `HScreen`.

Since:

MSM I01

setDisplayArea

```
void setDisplayArea(org.havi.ui.HScreenRectangle rect)
    throws java.lang.SecurityException,
```

`java.lang.IllegalStateException`

Set area of display screen to which logical screen is mapped.

Associates this logical HScreen with an area (extent) of its associated display HScreen.

Parameters:

`rect` - an `HScreenRectangle` instance specifying the area on the display HScreen associated with this logical HScreen that SHALL correspond to the extent of this logical HScreen.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalStateException` - if (1) this HScreen's type is not `SCREEN_TYPE_LOGICAL` or (2) the area of the display HScreen associated with this logical HScreen cannot be changed, e.g., if the platform uses a permanent association with a specific area of the associated display HScreen.

Since:

MSM I01

checkServiceContextCompatibility

`boolean`

`checkServiceContextCompatibility`(`javax.tv.service.selection.ServiceContext context`)

throws `java.lang.SecurityException`,

`java.lang.IllegalArgumentException`

Test compatibility of service context with screen.

Determine if application may assign `ServiceContext` to this HScreen and if the specified `ServiceContext` is compatible with presentation on this HScreen.

Parameters:

`context` - a valid `ServiceContext` instance.

Returns:

A boolean value indicating if the specified `ServiceContext` instance can be assigned to this HScreen. If it can be assigned, `true` is returned; otherwise, `false` is returned.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalArgumentException` - if the specified `ServiceContext` is not valid.

Since:

MSM I01

addServiceContexts

`void addServiceContexts`(`javax.tv.service.selection.ServiceContext[] contexts`,
`boolean associateDefaultDevices`)
 throws `java.lang.SecurityException`,
`java.lang.IllegalStateException`

Add service context(s) to this screen.

Add one or more `ServiceContext` instances to the set of service contexts associated with this HScreen in order to permit background, video, and graphics content from these `ServiceContext` instances to be presented on the HScreen's respective screen devices.

If a specified service context is already associated with the underlying screen represented by this interface, then that service context is not multiply associated with this screen, but the existing association remains

intact; that is, a given service context SHALL be associated with a given (underlying) screen either once or not at all.

If more than one non-abstract service context is associated with a given screen, and if multiple video sources from the background based players of these multiple non-abstract service contexts are associated with a given `HVideoDevice` instance of the given screen, then the background based player from these multiple service contexts whose video content is to be displayed is determined according to the following ordered rules:

1. If the owning application of one of the associated non-abstract service contexts holds a reservation on a given `HVideoDevice` of the screen, then background based player content from that service context is designated for presentation on that video device;
2. Otherwise, the background based player of the associated non-abstract service context whose application is assigned the highest priority is designated for presentation on that video device;

If, after applying the above rules, multiple background based players of a given application are designated for presentation on a video device, then the player that was most recently started is given priority for video presentation.

Parameters:

`contexts` - a non-empty array of `ServiceContext` instances.

`associateDefaultDevices` - if true, then associate default screen devices of this screen with all `ServiceMediaHandler` instances currently associated with the specified `ServiceContext` instances; otherwise, if false, then do not perform this association with default screen devices.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted

`MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalStateException` - if some specified `ServiceContext` for this `HScreen` cannot be changed, e.g., if the platform uses a permanent association with a specific `ServiceContext`.

Since:

MSM I01

removeServiceContexts

void

```
removeServiceContexts( javax.tv.service.selection.ServiceContext[] contexts)
                        throws java.lang.SecurityException,
                               java.lang.IllegalArgumentException,
                               java.lang.IllegalStateException
```

Remove service context(s) from screen.

Remove all or some non-empty set of specific `ServiceContext` instances from the set of service contexts associated with this `HScreen`. If `contexts` is null, then all service contexts are removed.

Parameters:

`contexts` - either null or a non-empty array of `ServiceContext` instances.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted

`MonitorAppPermission("multiscreen.context")`.

`java.lang.IllegalArgumentException` - if `contexts` is not null and some `ServiceContext` entry is not associated with this `HScreen` instance.

`java.lang.IllegalStateException` - if a specified `ServiceContext` cannot be changed, e.g., if the platform uses a permanent association with a specified `ServiceContext`.

Since:

MSM I01

setMultiScreenConfiguration

```
void setMultiScreenConfiguration(MultiScreenConfiguration configuration,
    java.util.Dictionary serviceContextAssociations)
    throws java.lang.SecurityException,
           java.lang.IllegalArgumentException,
           java.lang.IllegalStateException
```

Set currently active multiscreen configuration for this display screen (i.e., choose among the set of subsets of logical screens that may be associated with this display screen at a given time).

If the specified *configuration* is the current configuration for this display screen, then, unless *SecurityException* applies, return from this method without producing any side effect.

If the specified *configuration* is not the current configuration for this display screen and if *SecurityException*, *IllegalStateException*, and *IllegalStateException* do not apply, then perform the synchronous display multiscreen configuration change processing defined in the *OCAP Multiscreen Manager (MSM) Extension* specification.

If a *serviceContextAssociations* argument is specified (i.e., not null), then any *ServiceContext* instance that is accessible to the invoking application SHALL be associated with either no screen or the applicable screen(s) in the specified (new) multiscreen configuration. If no association matches some accessible *ServiceContext*, if some accessible *ServiceContext* instance is not present in the specified associations, or if it is present but no such applicable screen exists in the new multiscreen configuration, then the *ServiceContext* instance SHALL be associated with the default service context association screen of the specified multiscreen configuration, i.e., the screen returned by *configuration.getDefaultServiceContextScreen()*.

For the purpose of matching accessible *ServiceContext* instances whose references appear as keys in a specified *serviceContextAssociations* dictionary, the virtual method *equals(Object)* on these keys SHALL be used, in which case it is assumed that this method behaves identically to the default implementation of *Object.equals(Object)*.

Note: In the context of a given application instance, the MSM host implementation should maintain a one-to-one relationship between *ServiceContext* instances exposed to that application and collections of underlying service context resources. If the MSM host implementation fails to maintain this relationship, then when consulting a *serviceContextAssociations* dictionary, the MSM implementation may consider two distinct collections of underlying service context resources to be the same service context, e.g., if at different times, a single *ServiceContext* instance references distinct underlying collections of resources, or may consider a single collection of underlying service context resources to be two distinct service contexts, e.g., if at a given time, two distinct *ServiceContext* instances reference the same underlying collection of resources.

The state of the decoder format conversion (DFC) component of a video pipeline being used to process video associated with a service context that is implicitly swapped or moved between screens by this method SHALL NOT be affected by performance of this method.

Parameters:

configuration - a *MultiScreenConfiguration* instance to become the currently active per-display multiscreen configuration for this display screen.

serviceContextAssociations - if not null, then a *Dictionary* instance whose keys are *ServiceContext* instances and whose values are *String* instances, where the string values are defined as follows: (1) if the string value is "-", then no screen applies (in which case a matching service context is not associated with any screen after the configuration change), (2) otherwise, if the string value is "*", then all screens of the new screen configuration apply, (3) otherwise, if the string value is a screen identifier as returned by *MultiScreenContext.getID()*, then that screen applies, (4) otherwise, if the string value is a screen category as returned by

`MultiScreenContext.getScreenCategory()`, then any screen of the new configuration with that category applies, (5) otherwise, if the string value is a semicolon separated list of screen identifiers, then each screen of the new configuration with a matching identifier applies, (6) otherwise, if the string value is a semicolon separated list of screen categories, then each screen of the new configuration with a matching category applies, (7) otherwise, if the string value is a semicolon separated list of screen identifiers or screen categories, then each screen of the new configuration with a matching identifier or category applies, (8) otherwise, the default service context association screen of the new configuration applies.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted

`MonitorAppPermission("multiscreen.configuration")`.

`java.lang.IllegalArgumentException` - if *configuration* is not one of this (display) screen's multiscreen configurations as would be returned by

`MultiScreenContext.getMultiScreenConfigurations()`.

`java.lang.IllegalStateException` - if this screen is not a display screen or if the MSM implementation (1) does not permit activation of the specified multiscreen configuration, (2) if this method was previously called and the change processing steps are not yet complete, or (3) if activation is not otherwise permitted at the time of method invocation.

Since:

MSM I01

requestMultiScreenConfigurationChange

void

```
requestMultiScreenConfigurationChange(MultiScreenConfiguration configuration,
java.util.Dictionary serviceContextAssociations)
throws java.lang.SecurityException,
java.lang.IllegalArgumentException,
java.lang.IllegalStateException
```

Request that the currently active multiscreen configuration for this display screen be changed.

If the specified *configuration* is the current configuration for this display screen, then, unless `SecurityException` applies, return from this method without producing any side effect.

If the specified *configuration* is not the current display configuration and if `SecurityException`, `IllegalArgumentException`, and `IllegalStateException` do not apply, then initiate an asynchronous change to the current multiscreen configuration, where the semantics of `setMultiScreenConfiguration()` apply except that this method SHALL return immediately after `MultiScreenConfiguration.MULTI_SCREEN_CONFIGURATION_CHANGING` is generated (but before it is dispatched).

Parameters:

configuration - a `MultiScreenConfiguration` instance to become the currently active screen configuration.

serviceContextAssociations - either null or a `Dictionary` instance whose keys are `ServiceContext` instances and whose values are `String` instances, with semantics as defined by `setMultiScreenConfiguration(..)` above.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted

`MonitorAppPermission("multiscreen.configuration")`.

`java.lang.IllegalArgumentException` - if *configuration* is not one of this (display) screen's multiscreen configurations as would be returned by

`MultiScreenContext.getMultiScreenConfigurations()`.

`java.lang.IllegalStateException` - if this screen is not a display screen or (1) if the MSM implementation does not permit activation of the specified multiscreen configuration, (2) if this method was previously called and the change processing steps are not yet complete, or (3) if activation is not otherwise permitted at the time of method invocation.

Since:

MSM I01

getClient

```
org.davic.resources.ResourceClient getClient()
```

Obtain the resource client that currently holds the reservation on the underlying screen and screen resources associated with this `HScreen` instance.

Specified by:

`getClient` in interface `org.davic.resources.ResourceProxy`

Returns:

a `ResourceClient` instance or null if the underlying screen and screen resources associated with this `HScreen` are not reserved.

Since:

MSM I01

reserveScreen

```
boolean reserveScreen(org.davic.resources.ResourceClient client,
                      java.lang.Object requestData)
```

Atomically reserve underlying resources of this screen.

Reserving a screen SHALL be considered equivalent to reserving all `HScreenDevice` instances associated with the screen, except that when the screen is reserved using this method, individual `HScreenDevice` instances SHALL not be released without first releasing all `HScreenDevice` instances and all underlying screen resources of this screen.

If when reserving a screen, some `HScreenDevice` of the screen is already reserved by another application, then that reservation must be successfully released (by the MSM implementation) prior to granting reservation to the screen; and, if it is not or cannot be released, then an attempt to reserve the screen SHALL fail (i.e., this method returns `false`).

If when reserving a screen, some `HScreenDevice` of the screen is already reserved by the reserving application, then that reservation SHALL be implicitly subsumed by the successful reservation of the entire screen.

If an attempt to reserve a screen using this method succeeds, i.e., returns `true`, then the `getClient()` method of all `HScreenDevice` instances of the screen SHALL return the same value as the `getClient()` method defined by this interface (as implemented by the concrete implementation of `HScreen`).

If this screen was previously unreserved and invocation of this method results in it becoming reserved, then, prior to returning from this method, a `MultiScreenResourceEvent.MULTI_SCREEN_RESOURCE_SCREEN_RESERVED` event SHALL be generated.

If the calling application already holds a reservation on this screen and if the specified `client` and `requestData` arguments are identical to those specified when the calling application was previously granted the reservation, then this method SHALL have no side effect and return `true`. However, if the calling application already holds a reservation on this screen but either the specified `client` or `requestData` argument differs from those specified when the calling application was previously granted

the reservation, then (1) the specified *client* and *requestData* SHALL replace those previously specified, (2) the calling application SHALL retain the reservation, and (3) this method SHALL return `true`.

Parameters:

client - a `ResourceClient` instance.

requestData - either null or an `Object` instance that implements the `java.rmi.Remote` interface.

Returns:

`true` if underlying resources of screen were successfully reserved, otherwise `false`.

Since:

MSM I01

See Also:

`Remote`

releaseScreen

`void releaseScreen()`

Atomically release underlying resources of screen.

If the calling application does not hold a reservation on this screen, then this method SHALL have no side effect.

If this screen was previously reserved and invocation of this method results in it being no longer reserved (i.e., becoming unreserved), then, prior to returning from this method, a `MultiScreenResourceEvent.MULTI_SCREEN_RESOURCE_SCREEN_RELEASED` event SHALL be generated.

Since:

MSM I01

org.ocap.ui Interface MultiScreenConfiguration

public interface **MultiScreenConfiguration**

The **MultiScreenConfiguration** interface, implemented by an OCAP host platform, provides information on a discrete screen configuration as well as a mechanism for monitoring changes to the screen configuration.

An MSM implementation SHALL support at least one multiscreen configuration whose configuration type is **SCREEN_CONFIGURATION_DISPLAY**.

An MSM implementation SHALL support at least one multiscreen configuration whose configuration type is **SCREEN_CONFIGURATION_NON_PIP**.

If an MSM implementation implements PIP functionality and permits the presentation of a PIP screen during OCAP operation, then the MSM implementation SHALL support the multiscreen configuration **SCREEN_CONFIGURATION_PIP**.

If an MSM implementation implements POP functionality and permits the presentation of a POP screen during OCAP operation, then the MSM implementation SHALL support the multiscreen configuration **SCREEN_CONFIGURATION_POP**.

If an MSM implementation implements PIP, POP, or OVERLAY functionality in a discrete configuration not explicitly specified below by a non-general multiscreen configuration, then the MSM implementation SHALL support one or more multiscreen configurations of configuration type **SCREEN_CONFIGURATION_GENERAL** that represent each such multiscreen configuration.

Since:

MSM I01

Field Summary	
static java.lang.String	SCREEN_CATEGORY_DISPLAY If a display HScreen instance is characterized as a non-main screen, then its category is SCREEN_CATEGORY_DISPLAY .
static java.lang.String	SCREEN_CATEGORY_GENERAL If a logical HScreen instance is capable of being configured (e.g., in its size, position, or z-order) such that it may operate in a mode that would have suggested assignment of two or more of the other screen categories, then its category MAY be SCREEN_CATEGORY_GENERAL .
static java.lang.String	SCREEN_CATEGORY_MAIN If a display or logical HScreen instance is characterized as a main screen, then its category is SCREEN_CATEGORY_MAIN .
static java.lang.String	SCREEN_CATEGORY_NONE If an HScreen instance is not associated with any other more specific category, then its category is SCREEN_CATEGORY_NONE .
static java.lang.String	SCREEN_CATEGORY_OVERLAY If a logical HScreen instance is characterized as an overlay screen, then its category is SCREEN_CATEGORY_OVERLAY .

Field Summary	
static java.lang.String	SCREEN_CATEGORY_PIP If a logical HScreen instance is characterized as a picture-in-picture (PIP) screen, then its category is SCREEN_CATEGORY_PIP.
static java.lang.String	SCREEN_CATEGORY_POP If a logical HScreen instance is characterized as a picture-outside-picture (POP) screen, then its category is SCREEN_CATEGORY_POP.
static java.lang.String	SCREEN_CONFIGURATION_DISPLAY If a MultiScreenConfiguration is associated with one or more display screens and is a candidate for being used as a per-platform multiscreen configuration, then its configuration type is SCREEN_CONFIGURATION_DISPLAY.
static java.lang.String	SCREEN_CONFIGURATION_GENERAL If a MultiScreenConfiguration cannot be categorized as one of the other predefined screen configuration types, then its configuration type is SCREEN_CONFIGURATION_GENERAL.
static java.lang.String	SCREEN_CONFIGURATION_NON_PIP If a MultiScreenConfiguration is associated with exactly one screen whose category is SCREEN_CATEGORY_MAIN, then its configuration type is SCREEN_CONFIGURATION_NON_PIP.
static java.lang.String	SCREEN_CONFIGURATION_PIP If a MultiScreenConfiguration is (1) associated with one logical screen with default z-order of zero mapped to the entire area of a single display screen, and (2) associated with one or more non-intersecting logical screens with default z-order of one mapped to the same display screen, then its configuration type is SCREEN_CONFIGURATION_PIP.
static java.lang.String	SCREEN_CONFIGURATION_POP If a MultiScreenConfiguration is associated with two or more non-intersecting logical screens with default z-order of zero whose default display areas (in union) tile the entire area of a single display screen, then its configuration type is SCREEN_CONFIGURATION_POP.

Method Summary	
java.lang.String	getConfigurationType() Gets the screen configuration type of this configuration.
org.havi.ui.HScreen	getDefaultServiceContextScreen() Obtain the default service context association screen of this configuration.
org.havi.ui.HScreen[]	getScreens() Gets the set of accessible screens associated with this configuration.
org.havi.ui.HScreen[]	getScreens(java.lang.String category) Obtain all accessible screens with a given category.
boolean	hasOverlayScreen() Determine if the set of screens associated with this configuration includes an overlay screen, i.e., a screen whose category is SCREEN_CATEGORY_OVERLAY.

Method Summary

boolean	isScreenInConfiguration (org.havi.ui.HScreen screen) Determine if the underlying resources of a specified screen is represented by an HScreen instance included in the set of screens associated with this configuration.
void	setDefaultServiceContextScreen (org.havi.ui.HScreen screen) Set the default service context association screen of this configuration.

Field Detail

SCREEN_CONFIGURATION_DISPLAY

static final java.lang.String **SCREEN_CONFIGURATION_DISPLAY**

If a MultiScreenConfiguration is associated with one or more display screens and is a candidate for being used as a per-platform multiscreen configuration, then its configuration type is SCREEN_CONFIGURATION_DISPLAY.

The initial default screen of a SCREEN_CONFIGURATION_DISPLAY configuration SHALL be the first screen returned by getScreens(SCREEN_CATEGORY_MAIN), or, if there is no such categorized screen in the configuration, then the first screen returned by getScreens().

Since:
MSM I01

SCREEN_CONFIGURATION_NON_PIP

static final java.lang.String **SCREEN_CONFIGURATION_NON_PIP**

If a MultiScreenConfiguration is associated with exactly one screen whose category is SCREEN_CATEGORY_MAIN, then its configuration type is SCREEN_CONFIGURATION_NON_PIP.

The initial default screen of a SCREEN_CONFIGURATION_NON_PIP configuration SHALL be the first screen returned by getScreens(SCREEN_CATEGORY_MAIN).

A MultiScreenConfiguration instance that is categorized as having a SCREEN_CONFIGURATION_NON_PIP configuration type SHALL NOT contain more than one display screen.

Since:
MSM I01

SCREEN_CONFIGURATION_PIP

static final java.lang.String **SCREEN_CONFIGURATION_PIP**

If a MultiScreenConfiguration is (1) associated with one logical screen with default z-order of zero mapped to the entire area of a single display screen, and (2) associated with one or more non-intersecting logical screens with default z-order of one mapped to the same display screen, then its configuration type is SCREEN_CONFIGURATION_PIP.

The initial default screen of a SCREEN_CONFIGURATION_PIP configuration SHALL be the first screen returned by getScreens(SCREEN_CATEGORY_MAIN), or, if there is no such categorized screen in the configuration, then the first screen returned by getScreens().

A `MultiScreenConfiguration` instance that is categorized as having a `SCREEN_CONFIGURATION_PIP` configuration type SHALL NOT contain more than one display screen.

Since:
MSM I01

SCREEN_CONFIGURATION_POP

`static final java.lang.String SCREEN_CONFIGURATION_POP`

If a `MultiScreenConfiguration` is associated with two or more non-intersecting logical screens with default z-order of zero whose default display areas (in union) tile the entire area of a single display screen, then its configuration type is `SCREEN_CONFIGURATION_POP`.

The initial default screen of a `SCREEN_CONFIGURATION_POP` configuration SHALL be the first screen returned by `getScreens(SCREEN_CATEGORY_POP)`, or, if there is no such categorized screen in the configuration, then the first screen returned by `getScreens()`.

A `MultiScreenConfiguration` instance that is categorized as having a `SCREEN_CONFIGURATION_POP` configuration type SHALL NOT contain more than one display screen.

Since:
MSM I01

SCREEN_CONFIGURATION_GENERAL

`static final java.lang.String SCREEN_CONFIGURATION_GENERAL`

If a `MultiScreenConfiguration` cannot be categorized as one of the other predefined screen configuration types, then its configuration type is `SCREEN_CONFIGURATION_GENERAL`.

The initial default screen of a `SCREEN_CONFIGURATION_GENERAL` configuration SHALL be the first screen returned by `getScreens(SCREEN_CATEGORY_MAIN)`, or, if there is no such categorized screen in the configuration, then the first screen returned by `getScreens()`.

A `MultiScreenConfiguration` instance that is categorized as having a `SCREEN_CONFIGURATION_GENERAL` configuration type SHALL NOT contain more than one display screen.

Since:
MSM I01

SCREEN_CATEGORY_NONE

`static final java.lang.String SCREEN_CATEGORY_NONE`

If an `HScreen` instance is not associated with any other more specific category, then its category is `SCREEN_CATEGORY_NONE`.

A `MultiScreenConfiguration` instance that is categorized as having a `SCREEN_CONFIGURATION_NONE` configuration type SHALL NOT contain more than one display screen.

Since:
MSM I01

SCREEN_CATEGORY_DISPLAY

`static final java.lang.String SCREEN_CATEGORY_DISPLAY`

If a display HScreen instance is characterized as a non-main screen, then its category is SCREEN_CATEGORY_DISPLAY. A display screen assigned this category SHALL NOT be populated by an HBackgroundDevice or an HVideoDevice, but MAY be populated by one or more HGraphicsDevice instances that serve as overlays.

A display HScreen instance that is categorized as SCREEN_CATEGORY_DISPLAY has the exceptional property that its HGraphicsDevice instances, if any exist, SHALL overlay all logical screens mapped to the display screen. This property SHALL hold even though MultiScreenContext.getZOrder() returns 0 for any display screen.

Note: The exceptional property described above is intended to support (1) legacy device scenarios where a closed caption overlay appears over all other content, and (2) configurations where, rather than treating an overlay screen as a separate logical screen, an overlay screen is considered as an integral HGraphicsDevice of the display screen.

Since:
MSM I01

SCREEN_CATEGORY_MAIN

```
static final java.lang.String SCREEN_CATEGORY_MAIN
```

If a display or logical HScreen instance is characterized as a main screen, then its category is SCREEN_CATEGORY_MAIN. A logical screen assigned this category SHALL be mapped to the full area of some display screen and SHALL be assigned a default z-order of 0.

Since:
MSM I01

SCREEN_CATEGORY_PIP

```
static final java.lang.String SCREEN_CATEGORY_PIP
```

If a logical HScreen instance is characterized as a picture-in-picture (PIP) screen, then its category is SCREEN_CATEGORY_PIP. A logical screen assigned this category SHALL NOT be mapped to the full area of some display screen; SHALL NOT be assigned a screen z-order of 0; and SHALL co-exist in a configuration to which some screen is assigned the category SCREEN_CATEGORY_MAIN.

Since:
MSM I01

SCREEN_CATEGORY_POP

```
static final java.lang.String SCREEN_CATEGORY_POP
```

If a logical HScreen instance is characterized as a picture-outside-picture (POP) screen, then its category is SCREEN_CATEGORY_POP. A logical screen assigned this category SHALL NOT be mapped to the full area of some display screen; SHALL NOT co-exist in a configuration to which some screen is assigned the category SCREEN_CATEGORY_MAIN; and SHALL co-exist in a configuration to which some other screen is assigned the category SCREEN_CATEGORY_POP.

Since:
MSM I01

SCREEN_CATEGORY_OVERLAY

```
static final java.lang.String SCREEN_CATEGORY_OVERLAY
```

If a logical HScreen instance is characterized as an overlay screen, then its category is SCREEN_CATEGORY_OVERLAY. A logical screen assigned this category SHALL be mapped to the full area of some display screen; SHALL be assigned a default z-order greater than any screen associated with one of the following categories: SCREEN_CATEGORY_MAIN, SCREEN_CATEGORY_PIP, and

SCREEN_CATEGORY_POP; and SHALL NOT contain a background or an explicit video plane (HVideoDevice).

Notwithstanding the above, an overlay screen MAY make use of the resources of an implied video plane (HVideoDevice) for the purpose of presenting component based video in (one of) its graphics plane(s).

Since:
MSM I01

SCREEN_CATEGORY_GENERAL

static final java.lang.String **SCREEN_CATEGORY_GENERAL**

If a logical HScreen instance is capable of being configured (e.g., in its size, position, or z-order) such that it may operate in a mode that would have suggested assignment of two or more of the other screen categories, then its category MAY be SCREEN_CATEGORY_GENERAL. A logical screen assigned this category SHALL NOT be constrained in size, position, z-order, or other configurable properties except to the extent that the terminal device places intrinsic limitations on one (or more) configurable properties.

Since:
MSM I01

Method Detail

getConfigurationType

java.lang.String **getConfigurationType()**

Gets the screen configuration type of this configuration. If more than one non-general configuration type may apply or if the configuration type is unknown or cannot be determined, then the value SCREEN_CONFIGURATION_GENERAL SHALL be returned.

Returns:

A String that is either (1) an element of the enumeration { SCREEN_CONFIGURATION_DISPLAY, SCREEN_CONFIGURATION_NON_PIP, SCREEN_CONFIGURATION_PIP, SCREEN_CONFIGURATION_POP, SCREEN_CONFIGURATION_GENERAL }, or (2) a string value that denotes a platform-dependent configuration type and that starts with the prefix "x-".

Since:
MSM I01

getScreens

org.havi.ui.HScreen[] **getScreens()**

Gets the set of accessible screens associated with this configuration.

The underlying resources of a given HScreen instance returned by this method MAY be shared with an HScreen instance included in another multiscreen configuration; however, those shared resources SHALL be active in no more than one multiscreen configuration at a given time. The order of entries in the returned array of HScreen instances is implementation dependent.

Given the values of any two distinct entries of the returned array, *S1* and *S2*, and given the singleton instance of the MultiScreenManager, *MSM*, then the following constraints apply during the interval between the time when this method returns and the time when a MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING event is generated and dispatched:

- *MSM.isEmptyScreen(S1)* SHALL be false;
- *MSM.isEmptyScreen(S2)* SHALL be false;

- *MSM*.sameResources(*S1*,*S2*) SHALL be false;

If invoked by an application that does not have `MonitorAppPermission("multiscreen.configuration")` then the screens of this configuration that are not associated with a `ServiceContext` instance accessible by the application SHALL NOT be returned; otherwise, all accessible screens of this configuration SHALL be returned.

Over the course of an application's lifecycle, and except as constrained below, an MSM implementation MAY add screens to or remove screens from a multiscreen configuration, in which case it SHALL generate a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED` or `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED` event, respectively.

A screen SHALL NOT be added to or removed from a multiscreen configuration that is the currently active per-platform multiscreen configuration or some currently active per-display multiscreen configuration.

Note: The MSM implementation must wait until a multiscreen configuration is no longer the currently active multiscreen configuration in order to add or remove screens from that configuration.

During any time interval in which no `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED` or `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED` event is generated, the set of returned `HScreen` instances and the order of these returned instances SHALL not change from the perspective of a given application.

An MSM implementation SHALL not remove nor generate a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED` event that would have the effect of removing (or reporting the removal) from this multiscreen configuration an `HScreen` instance whose underlying resources represent the same underlying resources of some non-destroyed application's default `HScreen` instance.

Returns:

A (possible empty) `HScreen` array.

Since:

MSM I01

getScreens

`org.havi.ui.HScreen[] getScreens(java.lang.String category)`

Obtain all accessible screens with a given category.

This method, `getScreens(String)`, SHALL function identically to the method `getScreens()` except as follows:

- If *category* is not null, then return only those screens assigned the specified category, or, if no such screen exists, then return an empty `HScreen` array; otherwise, if *category* is null, then return the same value as `getScreens()`.

Parameters:

category - one of the following values: (1) null, (2) an element of the following enumeration: { `SCREEN_CATEGORY_NONE`, `SCREEN_CATEGORY_DISPLAY`, `SCREEN_CATEGORY_MAIN`, `SCREEN_CATEGORY_PIP`, `SCREEN_CATEGORY_POP`, `SCREEN_CATEGORY_OVERLAY`, `SCREEN_CATEGORY_GENERAL` }, or (3) a platform-dependent string value not pre-defined as a screen category but that MAY be returned by `getScreenCategory(HScreen)`.

Returns:

A (possibly empty) `HScreen` array.

Since:

MSM I01

hasOverlayScreen`boolean hasOverlayScreen()`

Determine if the set of screens associated with this configuration includes an overlay screen, i.e., a screen whose category is `SCREEN_CATEGORY_OVERLAY`.

Returns:

true if `getScreens(SCREEN_CATEGORY_OVERLAY)` would return a non-empty array; otherwise, returns false.

Since:

MSM I01

getDefaultServiceContextScreen

```
org.havi.ui.HScreen getDefaultServiceContextScreen()  
                                throws  
java.lang.SecurityException
```

Obtain the default service context association screen of this configuration.

The default service context association screen of a multiscreen configuration is the screen with which `ServiceContext` instances are associated when the configuration becomes active in case that no more specific information is available to determine how to associate a `ServiceContext` instance with a screen.

The following constraints apply:

1. if this multiscreen configuration is a per-platform display multiscreen configuration, then the default service context association screen SHALL be a screen associated with the per-display multiscreen configuration of some display screen associated with this multiscreen configuration;
2. otherwise, if this multiscreen configuration is a per-display multiscreen configuration, then the default service context association screen SHALL be a display screen or a logical screen associated with this multiscreen configuration.

Returns:

an `HScreen` instance that serves as the default service context screen for this configuration.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.configuration")`.

Since:

MSM I01

setDefaultServiceContextScreen

```
void setDefaultServiceContextScreen(org.havi.ui.HScreen screen)  
                                throws java.lang.SecurityException,  
                                       java.lang.IllegalArgumentException
```

Set the default service context association screen of this configuration.

The default service context association screen of a multiscreen configuration is the screen with which `ServiceContext` instances are associated when the configuration becomes active in case that no more specific information is available to determine how to associate a `ServiceContext` instance with a screen.

Parameters:

screen - an HScreen instance to be designated as the default service context association screen for this multiscreen configuration.

Throws:

java.lang.SecurityException - if the calling thread has not been granted

MonitorAppPermission("multiscreen.configuration").

java.lang.IllegalArgumentException - if the constraints specified above under

getDefaultServiceContextScreen() are not satisfied.

Since:

MSM I01

isScreenInConfiguration

boolean **isScreenInConfiguration**(org.havi.ui.HScreen screen)

Determine if the underlying resources of a specified screen is represented by an HScreen instance included in the set of screens associated with this configuration.

Parameters:

screen - an HScreen instance.

Returns:

true if the underlying resources specified screen is represented by an HScreen instance included in this configuration; otherwise, false.

Since:

MSM I01

org.ocap.ui

Interface MultiScreenContext

All Known Subinterfaces:

MultiScreenConfigurableContext

```
public interface MultiScreenContext
```

This interface provides a set of tools for accomplishing the following:

1. distinguishing between HScreen instances that are directly associated with a VideoOutputPort and those that are indirectly associated with a VideoOutputPort through a logical mapping process; i.e., discovering whether an HScreen is a display screen or a logical screen;
2. discovering the unique platform identifier of an HScreen instance's underlying resource;
3. discovering the category assigned to an HScreen within its containing MultiScreenConfiguration;
4. discovering the mapping of logical HScreens to display HScreens including the area (extent) on the display HScreen where the logical HScreen appears, its visibility, and its z-order;
5. discovering the z-order of HScreenDevices within an HScreen;
6. discovering the set of ServiceContexts associated with an HScreen;
7. discovering the association of display HScreens and corresponding VideoOutputPort instances;
8. discovering the set of HScreenDevices whose generated audio constitute the set of audio sources of an HScreen;
9. discovering the current audio focus assignment of a display HScreen;
10. obtaining notification of changes in state of this MultiScreenContext or certain changes in the HScreen instance that implements this interface;
11. obtaining notification of changes to the per-display multiscreen configuration of a display screen;
12. discovering the set of per-display multiscreen configurations that may be used with a display screen;
13. obtaining the currently active per-display multiscreen configuration of a display screen;

If an OCAP implementation does not support the *OCAP Multiscreen Manager (MSM) Extension* and does not otherwise support this interface, then an OCAP application MAY assume that the behavior of an HScreen is equivalent to an HScreen instance that does implement this interface, MultiScreenContext, whose methods would behave as follows:

- `getScreenType()` returns `SCREEN_TYPE_DISPLAY`;
- `getID()` returns a platform dependent (possibly empty) string;
- `getScreenCategory()` returns the string "main";
- `getVisible()` returns `true`;

- `getZOrder()` returns 0;
- `getZOrder(HScreenDevice device)` returns the array index of *device* in an array of `HScreenDevice` instances created by concatenating the ordered results of invoking on this `HScreen` instance the methods `HScreen.getHBackgroundDevices()`, then `HScreen.getHVideoDevices()`, then `HScreen.getHGraphicsDevices()`, or throws an `IllegalArgumentException` in case *device* is not present in this array;
- `getAudioSources()` returns an array of `HScreenDevice` instances created by concatenating the ordered results of invoking on this `HScreen` instance the methods `HScreen.getHBackgroundDevices()`, then `HScreen.getHVideoDevices()`, then `HScreen.getHGraphicsDevices()`;
- `getAudioFocus()` returns this `HScreen` instance;
- `getOutputPorts()` returns an array containing all `VideoOutputPorts` available on the platform;
- `getDisplayScreen()` returns this `HScreen` instance;
- `getDisplayArea()` returns `new HScreenRectangle(0,0,1,1)`;
- `getContexts()` returns an array containing all `ServiceContexts` that are accessible by the current application;

An MSM implementation SHALL support the `MultiScreenContext` interface on every `HScreen` instance.

Since:

MSM I01

Field Summary	
<code>static int</code>	<p>SCREEN_TYPE_DISPLAY</p> <p>If an <code>HScreen</code> is directly associated with a <code>VideoOutputPort</code> and the extent of the <code>HScreen</code> is mapped to the extent of the video raster produced from the <code>VideoOutputPort</code>, then the type of the <code>HScreen</code> is <code>SCREEN_TYPE_DISPLAY</code>, and is referred to as a display <code>HScreen</code>.</p>
<code>static int</code>	<p>SCREEN_TYPE_LOGICAL</p> <p>If an <code>HScreen</code> is not directly associated with a <code>VideoOutputPort</code> or the extent of the <code>HScreen</code> is mapped to a sub-region of the video raster produced from some <code>VideoOutputPort</code>, then the type of the <code>HScreen</code> is <code>SCREEN_TYPE_LOGICAL</code>, and is referred to as a logical <code>HScreen</code>.</p>

Method Summary	
<code>void</code>	<p>addMultiScreenConfigurationListener <code>(MultiScreenConfigurationListener listener)</code></p> <p>Add a listener to be notified upon the occurrence of multiscreen configuration events that apply to this screen in the case it is a display screen.</p>
<code>void</code>	<p>addScreenContextListener <code>(MultiScreenContextListener listener)</code></p> <p>Add screen context listener.</p>

Method Summary	
org.havi.ui.HScreen	getAudioFocus() Obtain the audio focus screen.
org.havi.ui.HScreenDevice[]	getAudioSources() Obtain audio sources of this screen.
org.havi.ui.HScreenRectangle	getDisplayArea() Obtain area of the display screen to which this screen is mapped.
org.havi.ui.HScreen	getDisplayScreen() Obtain display screen with which this screen is associated.
java.lang.String	getID() Obtain a platform dependent unique identifier for the underlying collection of screen resources denoted by this screen, where the scope for uniqueness is no smaller than the set of screens associated with the currently active per-platform multiscreen configuration and all active per-display multiscreen configurations.
MultiScreenConfiguration	getMultiScreenConfiguration() Obtain the currently active per-display multiscreen configuration for this display screen.
MultiScreenConfiguration[]	getMultiScreenConfigurations() Obtain set of all per-display multiscreen configurations currently associated with this display screen where the configuration type of any such multiscreen configuration SHALL NOT be SCREEN_CONFIGURATION_DISPLAY).
MultiScreenConfiguration[]	getMultiScreenConfigurations (java.lang.String screenConfigurationType) Obtain per-display multiscreen configurations of a specific type associated with this display screen.
org.ocap.hardware.VideoOutputPort[]	getOutputPorts() Obtain video ports to which screen is mapped.
java.lang.String	getScreenCategory() Obtain the screen category of this HScreen instance.
int	getScreenType() Obtain the type of this HScreen.
javax.tv.service.selection.ServiceContext[]	getServiceContexts() Obtain service contexts associated with this screen.
boolean	getVisible() Obtain screen visibility.
int	getZOrder() Obtain screen z-order.
int	getZOrder (org.havi.ui.HScreenDevice device) Obtain screen device z-order within this screen.
void	removeMultiScreenConfigurationListener (MultiScreenConfigurationListener listener) Remove a listener previously added to be notified upon the occurrence of multiscreen configuration events.

Method Summary

void	removeScreenContextListener (MultiScreenContextListener listener) Remove screen context listener.
------	--

Field Detail

SCREEN_TYPE_DISPLAY

static final int **SCREEN_TYPE_DISPLAY**

If an HScreen is directly associated with a VideoOutputPort and the extent of the HScreen is mapped to the extent of the video raster produced from the VideoOutputPort, then the type of the HScreen is SCREEN_TYPE_DISPLAY, and is referred to as a display HScreen.

Since:

MSM I01

SCREEN_TYPE_LOGICAL

static final int **SCREEN_TYPE_LOGICAL**

If an HScreen is not directly associated with a VideoOutputPort or the extent of the HScreen is mapped to a sub-region of the video raster produced from some VideoOutputPort, then the type of the HScreen is SCREEN_TYPE_LOGICAL, and is referred to as a logical HScreen. A logical HScreen MAY be associated with a display HScreen. If a logical HScreen is not associated with a display HScreen, then a visible or audible manifestation SHALL NOT be produced by any ServiceContext associated with the logical HScreen.

Note: A logical HScreen that is not associated with a display HScreen may be decoding and using content for some purpose other than presentation, e.g., it may be recording the content from a ServiceContext for future presentation

Since:

MSM I01

Method Detail

getScreenType

int **getScreenType()**

Obtain the type of this HScreen.

Returns:

An integer value denoted by SCREEN_TYPE_DISPLAY or SCREEN_TYPE_LOGICAL.

Since:

MSM I01

getID

java.lang.String **getID()**

Obtain a platform dependent unique identifier for the underlying collection of screen resources denoted by this screen, where the scope for uniqueness is no smaller than the set of screens associated with the currently active per-platform multiscreen configuration and all active per-display multiscreen configurations. It is implementation dependent whether the scope for screen identifier uniqueness includes other, non-active multiscreen configurations or not.

A screen identifier SHALL NOT be equal to any screen category returned by `getScreenCategory()`.

If *S1* and *S2* are instances of `HScreen` in the context of the implemented scope of uniqueness and `MultiScreenManager.sameResources(S1,S2)` returns `false`, then `((MultiScreenContext)S1).getID()` and `((MultiScreenContext)S2).getID()` SHALL NOT return the same (equivalent) string; conversely, if `MultiScreenManager.sameResources(S1,S2)` returns `true`, then `((MultiScreenContext)S1).getID()` and `((MultiScreenContext)S2).getID()` SHALL return the same (equivalent) string.

Returns:

A string value denoting the collection of underlying resources this `HScreen` instance represents in the implemented scope of uniqueness.

Since:

MSM I01

getScreenCategory

`java.lang.String getScreenCategory()`

Obtain the screen category of this `HScreen` instance.

Returns:

A String that is either (1) an element of the following enumeration of constants defined by `MultiScreenConfiguration`: { `SCREEN_CATEGORY_DISPLAY`, `SCREEN_CATEGORY_MAIN`, `SCREEN_CATEGORY_PIP`, `SCREEN_CATEGORY_POP`, `SCREEN_CATEGORY_OVERLAY`, `SCREEN_CATEGORY_GENERAL` }, or (2) a string value that denotes a platform-dependent screen category and that starts with the prefix "x-".

Since:

MSM I01

getVisible

`boolean getVisible()`

Obtain screen visibility.

Determine whether this `HScreen` is marked as visible for presentation on some display `HScreen`, where "visible" is defined as producing a raster signal to a `VideoOutputPort`, whether or not the `VideoOutputPort` is enabled or disabled. A display `HScreen` SHALL remain marked as visible. A logical `HScreen` MAY be visible or hidden (not visible).

Returns:

A boolean value indicating whether this `HScreen` is marked visible or not on some display `HScreen`.

Since:

MSM I01

getZOrder

`int getZOrder()`

Obtain screen z-order.

Determine the z-order of this `HScreen`. An display `HScreen` SHALL always return a z-order of zero. A logical `HScreen` MAY be assigned a z-order of 1 or greater, unless it is not associated with a display `HScreen`, in which case its z-order is -1.

A greater z-order denotes a more front-most (top-most) order among a set of `HScreen` instances.

Returns:

A value indicating the z-order of this HScreen or -1. If this HScreen is a logical HScreen that is not associated with a display HScreen, then -1 SHALL be returned.

Since:

MSM I01

getZOrder

```
int getZOrder(org.havi.ui.HScreenDevice device)
    throws java.lang.IllegalArgumentException
```

Obtain screen device z-order within this screen.

Determine the z-order of a specified HScreenDevice with an HScreen where the following constraints apply:

- if an HBackgroundDevice is present in this screen, then the z-order of the rear-most (bottom-most) HBackgroundDevice is zero;
- if no HBackgroundDevice is present in this screen and if an HVideoDevice is present in this screen, then the z-order of the rear-most (bottom-most) HVideoDevice is zero;
- if no HBackgroundDevice and no HVideoDevice is present in this screen and if an HGraphicsDevice is present in this screen, then the z-order of the rear-most (bottom-most) HGraphicsDevice is zero;
- the z-order of an HVideoDevice is greater than the z-order of any HBackgroundDevice in this screen;
- the z-order of an HGraphicsDevice is greater than the z-order of any HVideoDevice in this screen;

A greater z-order denotes a more front-most (top-most) order among a set of HScreenDevice instances within an HScreen instance.

Each distinct set of underlying screen devices represented as an HScreen instance constitutes a distinct z-ordering; i.e., given multiple HScreen instances representing distinct underlying screens, the set of z-order values assigned to the underlying screen device resources of these screens may reuse the same z-order indices.

Parameters:

device - an HScreenDevice that is associated with this screen.

Returns:

A non-negative value indicating the z-order of the specified HScreenDevice.

Throws:

`java.lang.IllegalArgumentException` - if *device* is not an HScreenDevice of this screen.

Since:

MSM I01

getAudioSources

```
org.havi.ui.HScreenDevice[] getAudioSources()
```

Obtain audio sources of this screen.

Obtain the set of HScreenDevice from which presented audio is selected (and mixed) for the purpose of audio presentation from this screen.

The default set of audio sources of a screen consists of all `HScreenDevice` instances association with the screen.

The order of entries in the array returned by this method is not defined by this specification and SHALL be considered implementation dependent.

Returns:

A reference to an (possibly empty) array of `HScreenDevice` instances, where each such instance contributes to a mixed, audio presentation from this screen, or, if this screen does not support mixed audio, then at most one entry will be present in the returned array.

Since:

MSM I01

getAudioFocus

`org.havi.ui.HScreen` **getAudioFocus()**

Obtain the audio focus screen.

The audio focus screen of this `HScreen` is determined according to the following ordered rules, where the first rule that applies is used and others are ignored:

1. If this `HScreen` is a logical screen, then apply the following ordered sub-rules:
 - a. If this logical `HScreen` is mapped to a display screen, then apply the following sub-rules:
 - i. If this logical `HScreen` is currently assigned audio focus in the context of its display screen, then this logical `HScreen` is returned.
 - ii. Otherwise (not currently assigned audio focus in its display screen), `null` is returned.
 - b. Otherwise (not mapped to a display screen), if this logical `HScreen` is directly mapped to a video output port, then this `HScreen` is returned.
 - c. Otherwise (not mapped to a display screen and not directly mapped to a video output port), `null` is returned.
2. Otherwise (this `HScreen` is a display screen), apply the following sub-rules:
 - a. If some logical screen that is mapped to this display screen is assigned audio focus, then that logical `HScreen` is returned;
 - b. Otherwise (no logical screen is mapped to this display screen or no logical screen mapped to this display screen is assigned audio focus), then return this display `HScreen`.

The audio focus screen of a display screen is the screen whose currently selected audio sources are assigned to be rendered on all (implied) audio presentation devices of all video output ports to which the display screen is mapped.

Returns:

an `HScreen` instance or `null` as described above.

Since:

MSM I01

getOutputPorts

`org.ocap.hardware.VideoOutputPort[] getOutputPorts()`

Obtain video ports to which screen is mapped.

Obtain the set of `VideoOutputPort` instances associated with this `HScreen`. If this `HScreen`'s type is `SCREEN_TYPE_DISPLAY`, then the `VideoOutputPort` instances associated with this display screen SHALL be returned. If this `HScreen`'s type is `SCREEN_TYPE_LOGICAL` and this `HScreen` is associated with a display `HScreen`, then the `VideoOutputPort` instances associated with that display `HScreen` SHALL be returned. If this `HScreen`'s type is `SCREEN_TYPE_LOGICAL` and this `HScreen` is not associated with a display `HScreen`, then an empty array SHALL be returned.

Returns:

A reference to an array of `VideoOutputPort` instances. If the returned array is empty, then this `HScreen` is not associated with any `VideoOutputPort`.

Since:

MSM I01

getDisplayScreen

`org.havi.ui.HScreen getDisplayScreen()`

Obtain display screen with which this screen is associated.

Obtain the display `HScreen` associated with this `HScreen`. If this `HScreen`'s type is `SCREEN_TYPE_DISPLAY`, then a reference to this `HScreen` SHALL be returned. If this `HScreen`'s type is `SCREEN_TYPE_LOGICAL` and this `HScreen` is associated with a display `HScreen`, then that display `HScreen` SHALL be returned. If this `HScreen`'s type is `SCREEN_TYPE_LOGICAL` and this `HScreen` is not associated with a display `HScreen`, then the value `null` SHALL be returned.

Returns:

A reference to a display `HScreen` instance or `null`. If `null`, then this `HScreen` is not associated with a display `HScreen`.

Since:

MSM I01

getDisplayArea

`org.havi.ui.HScreenRectangle getDisplayArea()`

Obtain area of the display screen to which this screen is mapped.

Obtain the area (extent) of this `HScreen`. If this `HScreen`'s type is `SCREEN_TYPE_DISPLAY`, then an `HScreenRectangle` whose value is equal to `HScreenRectangle(0,0,1,1)` SHALL be returned. If this `HScreen`'s type is `SCREEN_TYPE_LOGICAL` and this `HScreen` is associated with a display `HScreen`, then the area (extent) occupied by this logical `HScreen` on its associated display `HScreen` SHALL be returned. If this `HScreen`'s type is `SCREEN_TYPE_LOGICAL` and this `HScreen` is not associated with a display `HScreen`, then the value `null` SHALL be returned.

Returns:

A reference to an `HScreenRectangle` instance or `null`. If `null`, then this `HScreen` is not associated with a display `HScreen`.

Since:

MSM I01

getServiceContexts

`javax.tv.service.selection.ServiceContext[] getServiceContexts()`

Obtain service contexts associated with this screen.

Obtain the set of `ServiceContexts` associated with this `HScreen` to which the calling application is granted access.

Returns:

A reference to an array of `ServiceContext` instances. If the returned array is empty, then this `HScreen` is not associated with any accessible `ServiceContext`.

Since:

MSM I01

addScreenContextListener

```
void addScreenContextListener(MultiScreenContextListener listener)
```

Add screen context listener.

Add a listener to be notified upon the occurrence of screen context events. If a listener has previously been added and not subsequently removed, then an attempt to add it again SHALL NOT produce a side effect.

Parameters:

`listener` - a `MultiScreenContextListener` instance.

Since:

MSM I01

removeScreenContextListener

```
void removeScreenContextListener(MultiScreenContextListener listener)
```

Remove screen context listener.

Remove a listener previously added to be notified upon the occurrence of screen context events. If the specified listener is not currently registered as a listener, then an attempt to remove it SHALL NOT produce a side effect.

Parameters:

`listener` - a `MultiScreenContextListener` instance.

Since:

MSM I01

addMultiScreenConfigurationListener

```
void
```

```
addMultiScreenConfigurationListener(MultiScreenConfigurationListener listener)
                                     throws
```

```
java.lang.IllegalStateException
```

Add a listener to be notified upon the occurrence of multiscreen configuration events that apply to this screen in the case it is a display screen. If a listener has previously been added and not subsequently removed, then an attempt to add it again SHALL NOT produce a side effect.

Configuration events that apply to a display screen SHALL be restricted to those that affect the complement of logical screens associated with the display screen.

If an event defined by `MultiScreenConfigurationEvent` is generated, then the MSM implementation SHALL notify each registered screen configuration listener accordingly.

Parameters:

`listener` - a `MultiScreenConfigurationListener` instance.

Throws:

`java.lang.IllegalStateException` - if the type of this screen is not `SCREEN_TYPE_DISPLAY`.

Since:

MSM I01

removeMultiScreenConfigurationListener

void

removeMultiScreenConfigurationListener(MultiScreenConfigurationListener listener)

Remove a listener previously added to be notified upon the occurrence of multiscreen configuration events. If the specified listener is not currently registered as a listener, then an attempt to remove it SHALL NOT produce a side effect.

Parameters:

listener - a MultiScreenConfigurationListener instance.

Since:

MSM I01

getMultiScreenConfigurations

MultiScreenConfiguration[] **getMultiScreenConfigurations**()
throws

java.lang.SecurityException

Obtain set of all per-display multiscreen configurations currently associated with this display screen where the configuration type of any such multiscreen configuration SHALL NOT be SCREEN_CONFIGURATION_DISPLAY).

Returns:

A non-empty array of MultiScreenConfiguration instances.

Throws:

java.lang.SecurityException - if the calling thread has not been granted MonitorAppPermission("multiscreen.configuration").

Since:

MSM I01

getMultiScreenConfigurations

MultiScreenConfiguration[]
getMultiScreenConfigurations(java.lang.String screenConfigurationType)
throws

java.lang.SecurityException,

java.lang.IllegalArgumentException

Obtain per-display multiscreen configurations of a specific type associated with this display screen.

Parameters:

screenConfigurationType - (1) an element of the following enumeration of constants defined by MultiScreenConfiguration: { SCREEN_CONFIGURATION_NON_PIP, SCREEN_CONFIGURATION_PIP, SCREEN_CONFIGURATION_POP, SCREEN_CONFIGURATION_GENERAL }, or (2) some other platform-dependent value not pre-defined as a multiscreen configuration type.

Returns:

An array of MultiScreenConfiguration instances or null, depending on whether specified configuration type is supported or not.

Throws:

java.lang.SecurityException - if the calling thread has not been granted MonitorAppPermission("multiscreen.configuration").

java.lang.IllegalArgumentException - if screenConfigurationType is SCREEN_CONFIGURATION_DISPLAY, is not defined, or is otherwise unknown to the platform implementation.

Since:
MSM I01

getMultiScreenConfiguration

MultiScreenConfiguration **getMultiScreenConfiguration()**
throws

java.lang.IllegalStateException

Obtain the currently active per-display multiscreen configuration for this display screen.

Returns:

The currently active per-display MultiScreenConfiguration instance that applies to this display screen.

Throws:

java.lang.IllegalStateException - if this HScreen is not a display screen.

Since:

MSM I01

org.ocap.ui
Class MultiScreenManager

```
java.lang.Object
└─ org.ocap.ui.MultiScreenManager
All Implemented Interfaces:
    org.davic.resources.ResourceServer

public abstract class MultiScreenManager
extends java.lang.Object
implements org.davic.resources.ResourceServer
```

The `MultiScreenManager` class is an abstract, singleton management class implemented by an OCAP host platform that provides multiscreen management services.

For other semantic constraints and behavior that apply, see the *OCAP Multiscreen Manager (MSM) Extension* specification.

Since:
 MSM I01

Constructor Summary	
protected	MultiScreenManager() Protected default constructor.

Method Summary	
void	addMultiScreenConfigurationListener (MultiScreenConfigurationListener listener) Add a listener to be notified upon the occurrence of multiscreen configuration events.
void	addPlayerScreenDevices (javax.media.Player player, org.havi.ui.HScreenDevice[] devices) Add screen device(s) to a media player.
void	addResourceStatusListener (org.davic.resources.ResourceStatusListener listener) Add resource status listener.
org.havi.ui.HScreen[]	findScreens (javax.tv.service.selection.ServiceContext context) Find accessible screen(s) associated with specific service context.
org.havi.ui.HScreen[]	getCompatibleScreens (org.ocap.hardware.VideoOutputPort port) Obtain the set of accessible screens that are compatible with an output port.
org.havi.ui.HScreen	getDefaultScreen() Obtain the default HScreen instance.
org.havi.ui.HScreen	getEmptyScreen() Obtain the singleton <i>empty HScreen</i> instance.

Method Summary	
static MultiScreenManager	getInstance() Gets the singleton instance of the MultiScreenManager.
MultiScreenConfiguration	getMultiScreenConfiguration() Obtain currently active per-platform display multiscreen configuration.
MultiScreenConfiguration	getMultiScreenConfiguration(org.havi.ui.HScreen screen) Obtain the multiscreen configuration of a specific screen.
MultiScreenConfiguration[]	getMultiScreenConfigurations() Obtain the set of all current multiscreen configurations supported by this platform, irrespective of their configuration type.
MultiScreenConfiguration[]	getMultiScreenConfigurations (java.lang.String screenConfigurationType) Obtain multiscreen configurations of a specific configuration type.
org.havi.ui.HScreen	getOutputPortScreen (org.ocap.hardware.VideoOutputPort port) Obtain the screen associated with an output port.
org.havi.ui.HScreenDevice[]	getPlayerScreenDevices (javax.media.Player player) Obtain the set of screen devices currently assigned for use by a (JMF) media player.
org.havi.ui.HScreen[]	getScreens() Obtain the set of accessible HScreen instances.
boolean	isEmptyScreen (org.havi.ui.HScreen screen) Determines if an instance of HScreen is equivalent, in terms of constraint satisfaction, to the <i>empty HScreen</i> .
boolean	isEmptyScreenDevice (org.havi.ui.HScreenDevice device) Determines if an instance of HScreenDevice is equivalent, in terms of constraint satisfaction, to the <i>empty HScreenDevice</i> of the specific sub-type.
void	moveServiceContexts (org.havi.ui.HScreen src, org.havi.ui.HScreen dst, javax.tv.service.selection.ServiceContext[] contexts) Atomically move a set of specific service context from one HScreen instance to another HScreen instance.
void	removeMultiScreenConfigurationListener (MultiScreenConfigurationListener listener) Remove a listener previously added to be notified upon the occurrence of multiscreen configuration events.
void	removePlayerScreenDevices (javax.media.Player player, org.havi.ui.HScreenDevice[] devices) Remove screen device(s) from a media player.
void	removeResourceStatusListener (org.davic.resources.ResourceStatusListener listener) Remove resource status listener.

Method Summary

void	requestMultiScreenConfigurationChange (MultiScreenConfiguration configuration, java.util.Dictionary serviceContextAssociations) Request change to the currently active per-platform display multiscreen configuration.
boolean	sameResources (org.havi.ui.HScreenDevice device1, org.havi.ui.HScreenDevice device2) Determines if two HScreenDevice instances represent the same underlying platform resources and underlying resource state, i.e., are equivalent with respect to these underlying resources.
boolean	sameResources (org.havi.ui.HScreen screen1, org.havi.ui.HScreen screen2) Determines if two HScreen instances represent the same underlying platform resources and underlying resource state, i.e., are equivalent with respect to these underlying resources.
boolean	sameResources (javax.tv.service.selection.ServiceContext sc1, javax.tv.service.selection.ServiceContext sc2) Determines if two ServiceContext instances represent the same underlying platform resources and underlying resource state, i.e., are equivalent with respect to these underlying resources.
void	setMultiScreenConfiguration (MultiScreenConfiguration configuration, java.util.Dictionary serviceContextAssociations) Set currently active per-platform display multiscreen configuration.
void	swapServiceContexts (org.havi.ui.HScreen screen1, org.havi.ui.HScreen screen2, javax.tv.service.selection.ServiceContext[] exclusions) Atomically swap service contexts between two HScreen instances.

Methods inherited from class java.lang.Object

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

Methods inherited from interface org.davic.resources.ResourceServer

addResourceStatusEventListener, removeResourceStatusEventListener

Constructor Detail

MultiScreenManager

protected **MultiScreenManager**()
Protected default constructor.
Since:
MSM I01

Method Detail

getScreens

```
public org.havi.ui.HScreen[] getScreens()
```

Obtain the set of accessible HScreen instances.

The set of HScreen instances returned SHALL be determined as follows: when called by an OCAP application that is granted `MonitorAppPermission("multiscreen.configuration")`, then an HScreen instance SHALL be returned for each display screen and each logical screen exposed through any accessible `MultiScreenConfiguration` instance at the time of this method's invocation; otherwise, an HScreen instance SHALL be returned for each display screen and each logical screen with which an accessible `ServiceContext` is associated (either directly or indirectly) for the purpose of presentation or potential presentation.

The first HScreen instance in the returned `HScreen[]` array SHALL be by the same value returned by the `getDefaultScreen()` method of this class. Subsequent elements of the returned array do not follow a prescribed order, and an application SHALL NOT rely upon the order of these subsequent elements.

The set of HScreen instances returned by this method MAY change over the course of an application's lifecycle, such as when a screen is added to or removed from an accessible multiscreen configuration; however, the HScreen instance reference that represents an application's default HScreen SHALL remain constant over the application's lifecycle. That is, from the perspective of a given application instance, an MSM implementation SHALL always return the same HScreen instance reference as the first element of the returned array of HScreen instances.

Any HScreen instance returned by the this method SHALL NOT be equivalent to the *empty HScreen* during the interval between the time when the method returns and the time when a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING` or `MultiScreenContextEvent.MULTI_SCREEN_CONTEXT_SERVICE_CONTEXT_CHANGED` event is generated and dispatched.

Each HScreen instance returned by this method MAY return different sets of `HScreenDevice` instances from its `getHBackgroundDevices()`, `getHVideoDevices()`, and `getHGraphicsDevices()` methods over the course of an application's lifecycle; however, as described below under the definition of the `getDefaultScreen()` method, the set of default `HBackgroundDevice`, `HVideoDevice`, and `HGraphicsDevice` instances returned via these methods from a default HScreen instance SHALL remain constant over an application's lifecycle, while the underlying device resources and configurations of these default `HScreenDevice` instances MAY change.

Any background, video, or graphics screen device, i.e., `HBackgroundDevice`, `HVideoDevice`, or `HGraphicsDevice`, of any HScreen returned by this method SHALL NOT be equivalent to the *empty HScreenDevice* of its specific sub-type during the interval between the time when this method returns and the time when a `MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING` or `MultiScreenContextEvent.MULTI_SCREEN_CONTEXT_SERVICE_CONTEXT_CHANGED` event is generated.

The number of `HScreenDevice` instances returned from the `getHBackgroundDevices()`, `getHVideoDevices()`, and `getHGraphicsDevices()` methods of an HScreen instance returned by this method MAY change over the lifecycle of that HScreen instance. If they do change, then a `MultiScreenContextEvent.MULTI_SCREEN_CONTEXT_DEVICES_CHANGED` SHALL be generated and dispatched to all registered `MultiScreenContextListener` instances.

If the number of `HScreenDevice` instances of a particular type increases or remains the same (for a given `HScreen` instance), then any reference to a non-default `HScreenDevice` instance previously returned to the application SHALL remain viable, and SHALL, at the option of the MSM implementation, either (1) be reused to represent the new underlying device resources of that type of screen device or (2) be reset to the *empty HScreenDevice* of the appropriate sub-type as defined above. In the former case, the reused `HScreenDevice` instance SHALL be present in the set of screen devices returned from `getHBackgroundDevices()`, `getHVideoDevices()`, or `getHGraphicsDevices()` method according to its screen device type; in the latter case, the `HScreenDevice` instance whose state is reset to the *empty HScreenDevice* state SHALL NOT be present in the set of screen devices returned by these methods.

If the number of `HScreenDevice` instances of a particular type decreases (for a given `HScreen` instance), then any reference to a non-default `HScreenDevice` instance previously returned to the application SHALL remain viable, SHALL be reset to the *empty screen device state*, and SHALL NOT be present in the set of screen devices returned by the `HScreen` instance.

The net effect of the above specified behavior is that an application that accesses only its default `HScreen` and default `HScreenDevice` instances can continue to access and use those instances without any knowledge of the existence of MSM functionality. In contrast, an application that accesses non-default `HScreen` instances or non-default `HScreenDevice` instances needs to monitor changes to the current per-platform and per-display multiscreen configurations as well as changes to the set of screen devices associated with a non-default screen.

If a non-default `HScreen` instance that was previously referenced by an application is reset to the empty screen state as a result of a multiscreen configuration change, the application can detect this fact by comparing the set of `HScreen` instance references that were obtained prior to an appropriate `MultiScreenConfigurationEvent` with those obtainable after the event. After this event, those `HScreen` instances that were reset to the empty state will no longer be present in the array of `HScreen` instances returned by this method. Furthermore, those previously obtained `HScreen` instances can be queried as to whether they were reset by using the `isEmptyScreen(HScreen)` method defined by this class. Lastly, if the application continues to make use of such a reset `HScreen` instance, then its behavior is well defined and immutable.

Similarly, if a non-default `HScreenDevice` instance that was previously referenced by an application is reset to the empty screen device state as a result of a multiscreen configuration change, the application can detect this fact by comparing the set of `HScreenDevice` instance references that were obtained prior to an appropriate `MultiScreenContextEvent` with those obtainable after the event. After this event, those `HScreenDevice` instances that were reset to the empty state will no longer be accessible through set of accessible `HScreen` instances returned by this method. Furthermore, they can be queried as to whether they were reset by using the `isEmptyScreenDevice(HScreenDevice)` method defined by this class. Lastly, if the application continues to make use of such a reset `HScreenDevice` instance, then its behavior is well defined and immutable.

If an `HScreen` instance, *S*, does not implement the `MultiScreenConfigurableContext` interface (e.g., because it was not configurable when created and the MSM implementation selectively implements this interface on specific instances of `HScreen`) and if the new underlying screen resources are configurable (and thus this interface would be implemented on an `HScreen` instance that represents this set of underlying screen resources), then the MSM implementation SHALL NOT reuse *S* to represent a new set of underlying screen resources upon a multiscreen configuration change, but SHALL instead reset *S* to the empty state.

Returns:

A non-empty array of `HScreen` instances as described above.

Since:

MSM I01

getDefaultScreen

```
public org.havi.ui.HScreen getDefaultScreen()
```

Obtain the default HScreen instance.

The HScreen instance returned by this method SHALL represent the currently active set of default, underlying screen devices and their currently active HAVi screen configurations. In addition, it MAY represent a set of currently active non-default, underlying screen devices and their configurations.

The returned default HScreen instance is intended to be the **default** from the perspective of the calling application or the application on whose behalf this method is invoked. If invoked by the platform implementation in a context that does not imply a specific application, then the returned value is not defined and SHALL be implementation dependent, including, e.g., returning null.

Over the course of an application's lifecycle, the reference returned from this method SHALL remain constant. Furthermore, the set of default HScreenDevice instances returned by the getDefaultHBackgroundDevice(), getDefaultHVideoDevice(), and getDefaultHGraphicsDevice() methods of this default HScreen instance SHALL similarly remain constant over the application's lifecycle. Notwithstanding this constancy of reference, the underlying device resources and the underlying configurations of these device resources MAY change over the course of an application's lifecycle.

Any non-default background, video, or graphics screen device, i.e., HBackgroundDevice, HVideoDevice, or HGraphicsDevice, of any HScreen returned by this method SHALL NOT be equivalent to the *empty HScreenDevice* of its specific sub-type during the interval between the time when this method returns and the time when a MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING or MultiScreenContextEvent.MULTI_SCREEN_CONTEXT_SERVICE_CONTEXT_CHANGED event is generated.

If any HScreenConfiguration (or derived screen device type specific) parameter of any of the default HScreenDevice instances returned by the above cited methods changes over the course of an application's lifecycle, then an appropriate HScreenConfigurationEvent SHALL be generated and dispatched to all registered HScreenConfigurationListener instances. Similarly, if any value that would be returned by any of the query (*get**) methods of the MultiScreenContext interface implemented by this default HScreen instance changes, then an appropriate MultiScreenContextEvent SHALL be generated and dispatched to all registered MultiScreenContextListener instances.

If any HScreen instance returned by this method implements the MultiScreenConfigurableContext interface, then every HScreen instance returned by this method SHALL implement the MultiScreenConfigurableContext interface irrespective of whether the underlying screen resources represented by any returned HScreen instance is configurable or not.

Returns:

An HScreen instance as described above.

Since:

MSM I01

findScreens

```
public org.havi.ui.HScreen[]  
findScreens(javax.tv.service.selection.ServiceContext context)
```

Find accessible screen(s) associated with specific service context. A given service context may be associated with zero, one, or multiple HScreen instances.

Find the set of accessible HScreen instances with which a specified ServiceContext is associated. An HScreen instance is accessible (by some application) if the getScreens() method returns (or would return) that instance (when invoked by the same application at the time this method is invoked).

Parameters:

context - a ServiceContext instance.

Returns:

An array of HScreen instances or null. If the specified ServiceContext is associated with some accessible HScreen, then that HScreen SHALL be present in the returned array which SHALL be non-null and non-empty; otherwise, null is returned, indicating that the ServiceContext is not associated with any HScreen.

Since:

MSM I01

getOutputPortScreen

```
public org.havi.ui.HScreen
```

```
getOutputPortScreen(org.ocap.hardware.VideoOutputPort port)
```

Obtain the screen associated with an output port.

Given a specific VideoOutputPort instance, obtain the HScreen instance that is associated with that port for the purpose of presentation.

Parameters:

port - a VideoOutputPort instance.

Returns:

The HScreen instance associated with the specified port or null if no association exists.

Since:

MSM I01

getCompatibleScreens

```
public org.havi.ui.HScreen[]
```

```
getCompatibleScreens(org.ocap.hardware.VideoOutputPort port)
```

Obtain the set of accessible screens that are compatible with an output port.

Given a specific VideoOutputPort instance, obtain the subset of HScreen instances (of the larger set returned by getScreens()) that may be associated with that port for the purpose of presentation.

Parameters:

port - a VideoOutputPort instance.

Returns:

A (possibly empty) array of HScreen instances that may be associated with the specified port.

Since:

MSM I01

getMultiScreenConfigurations

```
public MultiScreenConfiguration[] getMultiScreenConfigurations()
throws
```

```
java.lang.SecurityException
```

Obtain the set of all current multiscreen configurations supported by this platform, irrespective of their configuration type.

The set of multiscreen configuration instances returned by this method SHALL include all per-platform multiscreen configurations (composed solely of display screens) and all per-display multiscreen configurations (composed of no more than one display screen and any number of logical screens).

The order of multiscreen configurations returned by this method is not defined by this specification.

Returns:

An array of `MultiScreenConfiguration` instances.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.configuration")`.

Since:

MSM I01

getMultiScreenConfigurations

```
public MultiScreenConfiguration[]
getMultiScreenConfigurations(java.lang.String screenConfigurationType)
                                throws
```

```
java.lang.SecurityException
```

Obtain multiscreen configurations of a specific configuration type.

Parameters:

`screenConfigurationType` - one of the following values: (1) an element of the following enumeration of constants defined by `MultiScreenConfiguration`: { `SCREEN_CONFIGURATION_DISPLAY`, `SCREEN_CONFIGURATION_NON_PIP`, `SCREEN_CONFIGURATION_PIP`, `SCREEN_CONFIGURATION_POP`, `SCREEN_CONFIGURATION_GENERAL` }, or (2) some other platform-dependent value not pre-defined as a multiscreen configuration type.

The set of multiscreen configuration instances returned by this method SHALL include all multiscreen configurations of the specified type that appear in the array returned by `getMultiScreenConfigurations()`.

The order of multiscreen configurations returned by this method is not defined by this specification.

Returns:

An array of `MultiScreenConfiguration` instances or null, depending on whether the specified configuration type is supported or not.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.configuration")`.

Since:

MSM I01

getMultiScreenConfiguration

```
public MultiScreenConfiguration
getMultiScreenConfiguration(org.havi.ui.HScreen screen)
```

Obtain the multiscreen configuration of a specific screen.

A given `HScreen` instance SHALL be associated with either zero or exactly one multiscreen configuration instance; however, since a single underlying screen MAY be potentially shared (multiply referenced) by multiple `HScreen` instances, an underlying screen (and its constituent resources) MAY be associated with more than one multiscreen configuration.

Parameters:

`screen` - an `HScreen` instance

Returns:

The `MultiScreenConfiguration` instance of which the specified `HScreen` is a constituent screen or null if the specified `HScreen` is orphaned (i.e., not owned by a multiscreen configuration, e.g., as would be the case if it were an empty screen).

Since:

MSM I01

getMultiScreenConfiguration

```
public MultiScreenConfiguration getMultiScreenConfiguration()
```

Obtain currently active per-platform display multiscreen configuration.

Returns:

The currently active per-platform display `MultiScreenConfiguration` instance.

Since:

MSM I01

setMultiScreenConfiguration

```
public void
```

```
setMultiScreenConfiguration(MultiScreenConfiguration configuration,
```

```
java.util.Dictionary serviceContextAssociations)
```

```
throws java.lang.SecurityException,  
        java.lang.IllegalStateException,  
        java.lang.IllegalStateException
```

Set currently active per-platform display multiscreen configuration.

If the specified *configuration* is the current per-platform display multiscreen configuration, then, unless `SecurityException` applies, return from this method without producing any side effect.

If the specified *configuration* is not the current per-platform display multiscreen configuration and if `SecurityException`, `IllegalArgumentException`, and `IllegalStateException` do not apply, then perform the synchronous per-platform multiscreen configuration change processing defined by the *OCAP Multiscreen Manager (MSM) Extension* specification.

If the *serviceContextAssociations* argument is specified (i.e., not null), then any `ServiceContext` instance that is accessible to the invoking application SHALL be associated with either no screen or the applicable screen(s) in the specified (new) per-platform multiscreen configuration (or in its per-display multiscreen configurations). If no association matches some accessible `ServiceContext`, if some accessible `ServiceContext` instance is not present in the specified associations, or if it is present but no such applicable screen exists in the new per-platform multiscreen configuration (or in its per-display multiscreen configurations), then the `ServiceContext` instance SHALL be associated with the default service context association screen of the specified multiscreen configuration, i.e., the screen returned by `configuration.getDefaultServiceContextScreen()`.

For the purpose of matching accessible `ServiceContext` instances whose references appear as keys in a specified *serviceContextAssociations* dictionary, the virtual method `equals(Object)` on these keys SHALL be used, in which case it is assumed that this method behaves identically to the default implementation of `Object.equals(Object)`.

Note: In the context of a given application instance, the MSM host implementation should maintain a one-to-one relationship between `ServiceContext` instances exposed to that application and collections of underlying service context resources. If the MSM host implementation fails to maintain this relationship, then when consulting a *serviceContextAssociations* dictionary, the MSM implementation may consider two distinct collections of underlying service context resources to be the same service context, e.g., if at different times, a single `ServiceContext` instance references distinct underlying collections

of resources, or may consider a single collection of underlying service context resources to be two distinct service contexts, e.g., if at a given time, two distinct `ServiceContext` instances reference the same underlying collection of resources.

The state of the decoder format conversion (DFC) component of a video pipeline being used to process video associated with a service context that is implicitly swapped or moved between screens by this method SHALL NOT be affected by performance of this method.

Parameters:

`configuration` - a `MultiScreenConfiguration` instance to become the currently active per-platform display multiscreen configuration.

`serviceContextAssociations` - if not null, then a `Dictionary` instance whose keys are `ServiceContext` instances and whose values are `String` instances, where the string values are defined as follows: (1) if the string value is "-", then no screen applies (in which case a matching service context is not associated with any screen after the configuration change), (2) otherwise, if the string value is "*", then all screens of the new screen configuration apply, (3) otherwise, if the string value is a screen identifier as returned by `MultiScreenContext.getID()`, then that screen applies, (4) otherwise, if the string value is a screen category as returned by

`MultiScreenContext.getScreenCategory()`, then any screen of the new configuration with that category applies, (5) otherwise, if the string value is a semicolon separated list of screen identifiers, then each screen of the new configuration with a matching identifier applies, (6) otherwise, if the string value is a semicolon separated list of screen categories, then each screen of the new configuration with a matching category applies, (7) otherwise, if the string value is a semicolon separated list of screen identifiers or screen categories, then each screen of the new configuration with a matching identifier or category applies, (8) otherwise, the default service context association screen of the new configuration applies.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.configuration")`.

`java.lang.IllegalArgumentException` - if `configuration` is not a per-platform multiscreen configuration that would be returned by `MultiScreenManager.getMultiScreenConfigurations(SCREEN_CONFIGURATION_DISPLAY)`.

`java.lang.IllegalStateException` - if the MSM implementation (1) does not permit activation of the specified multiscreen configuration, (2) if this method was previously called and the change processing steps are not yet complete, or (3) if activation is not otherwise permitted at the time of method invocation.

Since:

MSM I01

requestMultiScreenConfigurationChange

```
public void
requestMultiScreenConfigurationChange(MultiScreenConfiguration configuration,
java.util.Dictionary serviceContextAssociations)
    throws java.lang.SecurityException,
```

```
java.lang.IllegalStateException
```

Request change to the currently active per-platform display multiscreen configuration.

If the specified `configuration` is the current configuration, then, unless `SecurityException` applies, return from this method without producing any side effect.

If the specified `configuration` is not the current per-platform display multiscreen configuration and if `SecurityException`, `IllegalArgumentException`, and `IllegalStateException` do not

apply, then initiate an asynchronous change to the current per-platform multiscreen configuration, where the semantics of `setMultiScreenConfiguration()` apply except that this method SHALL return immediately after `MultiScreenConfiguration.MULTI_SCREEN_CONFIGURATION_CHANGING` is generated (but before it is dispatched).

Parameters:

`configuration` - a `MultiScreenConfiguration` instance to become the currently active screen configuration.
`serviceContextAssociations` - either null or a `Dictionary` instance whose keys are `ServiceContext` instances and whose values are `String` instances, with semantics as defined by `setMultiScreenConfiguration(..)` above.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted `MonitorAppPermission("multiscreen.configuration")`.
`java.lang.IllegalStateException` - (1) if the MSM implementation does not permit activation of the specified multiscreen configuration, (2) if this method was previously called and the change processing steps are not yet complete, or (3) if activation is not otherwise permitted at the time of method invocation.

Since:

MSM I01

addMultiScreenConfigurationListener

```
public void
addMultiScreenConfigurationListener(MultiScreenConfigurationListener listener)
```

Add a listener to be notified upon the occurrence of multiscreen configuration events. If a listener has previously been added and not subsequently removed, then an attempt to add it again SHALL NOT produce a side effect.

Configuration events that apply to this `MultiScreenManager` singleton instance SHALL be restricted to those that affect the complement of usable display screens.

If an event defined by `MultiScreenConfigurationEvent` is generated, then the MSM implementation SHALL notify each registered screen configuration listener accordingly.

Parameters:

`listener` - a `MultiScreenConfigurationListener` instance.

Since:

MSM I01

removeMultiScreenConfigurationListener

```
public void
removeMultiScreenConfigurationListener(MultiScreenConfigurationListener listener)
```

Remove a listener previously added to be notified upon the occurrence of multiscreen configuration events. If the specified listener is not currently registered as a listener, then an attempt to remove it SHALL NOT produce a side effect.

Parameters:

`listener` - a `MultiScreenConfigurationListener` instance.

Since:

MSM I01

addResourceStatusListener

```
public void
addResourceStatusListener(org.davic.resources.ResourceStatusListener listener)
```

Add resource status listener.

Parameters:
listener - a ResourceStatusListener instance.

Since:
MSM I01

removeResourceStatusListener

```
public void
removeResourceStatusListener(org.davic.resources.ResourceStatusListener listener)
```

Remove resource status listener.

Parameters:
listener - a ResourceStatusListener instance.

Since:
MSM I01

swapServiceContexts

```
public void swapServiceContexts(org.havi.ui.HScreen screen1,
                                org.havi.ui.HScreen screen2,
                                javax.tv.service.selection.ServiceContext[] exclusions)
                                throws java.lang.SecurityException,
                                       java.lang.IllegalStateException
```

Atomically swap service contexts between two HScreen instances.

This method is a convenience method for supporting the common function of swapping content presentation between screens. Similar results obtained by this method MAY also be accomplished by the more general mechanism of removing and adding service contexts to specific accessible screens by using the `MultiScreenConfigurableContext.addServiceContext(...)` and `MultiScreenConfigurableContext.removeServiceContext(...)` methods. Nevertheless, use of the more general method MAY result in more presentation transition artifacts than use of this method due to the atomic swap semantics of this method.

The state of the decoder format conversion (DFC) component of a video pipeline being used to process video associated with a service context that is being swapped by this method SHALL NOT be affected by performance of this method.

Parameters:

screen1 - an HScreen instance whose service contexts are to be swapped with those of *screen2*.
screen2 - an HScreen instance whose service contexts are to be swapped with those of *screen1*.
exclusions - if not null, then a non-empty array of ServiceContext instances to be excluded from the swap operation, i.e., whose screen associations are not to be affected by the swap.

Throws:

java.lang.SecurityException - if the calling thread has not been granted MonitorAppPermission("multiscreen.configuration").
java.lang.IllegalStateException - if any of the following hold: (1) video is being presented as component video rather than background video in either screen, or (2) the ServiceContexts for the specified screens cannot be changed, e.g., if the platform uses a permanent association with a specific ServiceContext and a screen.

Since:

MSM I01

moveServiceContexts

```
public void moveServiceContexts(org.havi.ui.HScreen src,
                                org.havi.ui.HScreen dst,
                                javax.tv.service.selection.ServiceContext[] contexts)
                                throws java.lang.SecurityException,
                                       java.lang.IllegalArgumentException,
                                       java.lang.IllegalStateException
```

Atomically move a set of specific service context from one HScreen instance to another HScreen instance.

This method is a convenience method for supporting the common function of moving content presentations between screens for a set of given service contexts. Similar results obtained by this method MAY also be accomplished by the more general mechanism of removing and adding the service context to specific accessible screens by using the

`MultiScreenConfigurableContext.addServiceContexts(...)` and `MultiScreenConfigurableContext.removeServiceContexts(...)` methods. Nevertheless, use of the more general method MAY result in more presentation transition artifacts than use of this method due to the atomic move semantics of this method.

The state of the decoder format conversion (DFC) component of a video pipeline being used to process video associated with a service context that is being moved by this method SHALL NOT be affected by performance of this method.

Parameters:

src - an HScreen instance from which the specified service contexts are to be moved.

dst - an HScreen instance to which the specified service contexts are to be moved.

contexts - a non-empty array of ServiceContext instances to be moved from *src* screen to *dst* screen.

Throws:

`java.lang.SecurityException` - if the calling thread has not been granted

`MonitorAppPermission("multiscreen.configuration")`.

`java.lang.IllegalArgumentException` - if some specified ServiceContext is not currently associated with the source HScreen instance, *src*.

`java.lang.IllegalStateException` - if any of the following hold: (1) video is being presented as component video rather than background video in either screen; (2) some specified ServiceContext for the specified screens cannot be moved, e.g., if the platform uses a permanent association with a specific ServiceContext and a screen; or (3) a non-abstract ServiceContext is already associated with the destination screen and the platform supports only one non-abstract ServiceContext per screen.

Since:

MSM I01

getPlayerScreenDevices

```
public org.havi.ui.HScreenDevice[]
getPlayerScreenDevices(javax.media.Player player)
    Obtain the set of screen devices currently assigned for use by a (JMF) media player.
```

Parameters:

player - a JMF Player instance to query for its set of screen devices.

Returns:

An array of HScreenDevice instances, which SHALL be empty if and only if there is no associated screen device.

Since:

MSM I01

addPlayerScreenDevices

```
public void addPlayerScreenDevices( javax.media.Player player,
                                     org.havi.ui.HScreenDevice[] devices)
    throws java.lang.SecurityException,
           java.lang.IllegalStateException
```

Add screen device(s) to a media player.

Parameters:

player - a JMF *Player* instance with which to associate the specified screen device(s).

devices - a non-empty array of *HScreenDevice* instances on which to present some type of rendered content from the specified media player.

Throws:

java.lang.SecurityException - if the calling thread has not been granted

MonitorAppPermission("multiscreen.context").

java.lang.IllegalStateException - if any of the following hold: (1) the specified *player* is not in a stopped state; (2) the specified screen *device* is not compatible with the specified *player*; (3) some underlying screen device of *devices* is not available for use by this application, e.g., due to being exclusively reserved by another application; or (4) some underlying screen device of *devices* is already associated with a media player, and that device does not support association with multiple media players.

Since:

MSM I01

See Also:

Player, *HScreenDevice*

removePlayerScreenDevices

```
public void removePlayerScreenDevices( javax.media.Player player,
                                         org.havi.ui.HScreenDevice[] devices)
    throws java.lang.SecurityException,
           java.lang.IllegalArgumentException,
           java.lang.IllegalStateException
```

Remove screen device(s) from a media player.

Removes all or a non-empty set of *HScreenDevice* instances from the set of screen devices on which the specified media player is presented (or otherwise associated for presentation). If *devices* is null, then all screen device associations are removed.

Parameters:

player - a JMF *Player* instance from which to remove association with the specified screen device.

devices - either null or a non-empty set of *HScreenDevice* instances to be disassociated from the specified media player.

Throws:

java.lang.SecurityException - if the calling thread has not been granted

MonitorAppPermission("multiscreen.context").

java.lang.IllegalArgumentException - if *devices* is not null and some entry of *devices* is not associated with the specified *Player* instance.

java.lang.IllegalStateException - if the specified *player* is not in a stopped state.

Since:

MSM I01

getEmptyScreen

```
public org.havi.ui.HScreen getEmptyScreen()
```

Obtain the singleton *empty HScreen* instance.

Using this *empty HScreen*, it is possible to obtain a reference to the *empty HScreenDevice*, *empty HScreenConfiguration*, and *empty HScreenConfigTemplate* of each available sub-type, e.g., *HBackgroundDevice*, *HVideoDevice*, *HGraphicsDevice*, etc.

Note: The presence of this method is primarily aimed at supporting the testing of MSM functionality, such as the semantics of `isEmptyScreen()`, `isEmptyScreenDevice()`, `sameResources()`, etc.

Returns:

the *empty HScreen* singleton

Since:

MSM I01

isEmptyScreen

```
public boolean isEmptyScreen(org.havi.ui.HScreen screen)
```

Determines if an instance of `HScreen` is equivalent, in terms of constraint satisfaction, to the *empty HScreen*.

If *screen* does not implement `MultiScreenConfigurableContext`, then those constraints that pertain to `MultiScreenConfigurableContext` SHALL be deemed to be equivalent.

Parameters:

screen - an `HScreen` instance.

Returns:

true if the specified *screen* is equivalent to the *empty HScreen*.

Since:

MSM I01

isEmptyScreenDevice

```
public boolean isEmptyScreenDevice(org.havi.ui.HScreenDevice device)
```

Determines if an instance of `HScreenDevice` is equivalent, in terms of constraint satisfaction, to the *empty HScreenDevice* of the specific sub-type.

Parameters:

device - an `HScreenDevice` instance of one of the following sub-types: `HBackgroundDevice`, `HVideoDevice`, or `HGraphicsDevice`.

Returns:

true if the specified *device* is equivalent to the *empty HScreenDevice* of the matching sub-type.

Since:

MSM I01

sameResources

```
public boolean sameResources(org.havi.ui.HScreen screen1,
                             org.havi.ui.HScreen screen2)
```

Determines if two `HScreen` instances represent the same underlying platform resources and underlying resource state, i.e., are equivalent with respect to these underlying resources.

For the purpose of determining equivalence, the following conditions SHALL apply:

- if exactly one of *screen1* and *screen2* is equivalent to the *empty HScreen*, then the two screens are not equivalent with respect to underlying resources;
- if, for each screen device, *BD1*, returned by `screen1.getHBackgroundDevices()` there is not exactly one screen device, *BD2*, returned by `screen2.getHBackgroundDevices()` such that `sameResources(BD1, BD2)` returns true, then the two screens are not equivalent with respect to underlying resources;

- if, for each screen device, *VD1*, returned by *screen1.getHVideoDevices()* there is not exactly one screen device, *VD2*, returned by *screen2.getHVideoDevices()* such that *sameResources(VD1,VD2)* returns *true*, then the two screens are not equivalent with respect to underlying resources;
- if, for each screen device, *GD1*, returned by *screen1.getHGraphicsDevices()* there is not exactly one screen device, *GD2*, returned by *screen2.getHGraphicsDevices()* such that *sameResources(GD1,GD2)* returns *true*, then the two screens are not equivalent with respect to underlying resources;
- if, given an equivalent set of template arguments, *screen1.getBestConfiguration(...)* and *screen2.getBestConfiguration(...)* would not return (nominally unordered) sets of equivalent *HScreenConfiguration* instances, then the two screens are not equivalent with respect to underlying resources;
- if, given an equivalent set of screen configuration arguments, *screen1.getCoherentScreenConfigurations(...)* and *screen2.getCoherentScreenConfigurations(...)* would not return sets of equivalent *HScreenConfiguration* instances, then the two screens are not equivalent with respect to underlying resources;
- if, given an equivalent set of screen configuration arguments, *screen1.setCoherentScreenConfigurations(...)* and *screen2.setCoherentScreenConfigurations(...)* would not return the same value or would not modify the set of screen devices associated with the specified screen configuration arguments such that those screen devices remain equivalent, then the two screens are not equivalent with respect to underlying resources;
- if none the above conditions apply, then *screen1* and *screen2* are deemed equivalent with respect to underlying resources;

Parameters:

screen1 - an *HScreen* instance.

screen2 - an *HScreen* instance.

Returns:

true if the specified screens are equivalent as described by the above conditions.

Since:

MSM I01

sameResources

```
public boolean sameResources(org.havi.ui.HScreenDevice device1,
                             org.havi.ui.HScreenDevice device2)
```

Determines if two *HScreenDevice* instances represent the same underlying platform resources and underlying resource state, i.e., are equivalent with respect to these underlying resources.

For the purpose of determining equivalence, the following conditions SHALL apply:

- if *device1* and *device2* are not of the same sub-type, *HBackgroundDevice*, *HVideoDevice*, or *HGraphicsDevice*, then the two screen devices are not equivalent with respect to underlying resources;
- if exactly one of *device1* and *device2* is equivalent to the *empty HScreenDevice* of the appropriate sub-type, then the two screen devices are not equivalent with respect to underlying resources;

- if `device1.getFlickerFilter()` and `device2.getFlickerFilter()` would not return the same value, then the two screen devices are not equivalent with respect to underlying resources;
- if `device1.getInterlaced()` and `device2.getInterlaced()` would not return the same value, then the two screen devices are not equivalent with respect to underlying resources;
- if `device1.getPixelAspectRatio()` and `device2.getPixelAspectRatio()` would not return equivalent values, then the two screen devices are not equivalent with respect to underlying resources;
- if `device1.getPixelResolution()` and `device2.getPixelResolution()` would not return equivalent values, then the two screen devices are not equivalent with respect to underlying resources;
- if `device1.getScreenArea()` and `device2.getScreenArea()` would not return equivalent values, then the two screen devices are not equivalent with respect to underlying resources;
- if given equivalent `HScreenConfiguration` instances, `SC1` and `SC2`, as arguments, `device1.getOffset(SC1)` and `device2.getOffset(SC2)` would not return equivalent values, then the two screen devices are not equivalent with respect to underlying resources;
- if given equivalent `HScreenConfiguration` instances, `SC1` and `SC2`, and equivalent `Point` instances, `P1` and `P2`, as arguments, `device1.convertTo(SC1, P1)` and `device2.convertTo(SC2, P2)` would not return equivalent values, then the two screen devices are not equivalent with respect to underlying resources;
- if `device1.getConfigurations()` and `device2.getConfigurations()` would not return (nominally unordered) sets of equivalent `HScreenConfiguration` instances, then the two screens are not equivalent with respect to underlying resources;
- if `device1.getCurrentConfiguration()` and `device2.getCurrentConfiguration()` would not return equivalent `HScreenConfiguration` instances, then the two screens are not equivalent with respect to underlying resources;
- if `device1.getDefaultConfiguration()` and `device2.getDefaultConfiguration()` would not return equivalent `HScreenConfiguration` instances, then the two screens are not equivalent with respect to underlying resources;
- if, given an equivalent template arguments or sets of template arguments, `device1.getBestConfiguration(...)` and `device2.getBestConfiguration(...)` would not return sets of equivalent `HScreenConfiguration` instances, then the two screens are not equivalent with respect to underlying resources;
- if `device1` and `device2` are instances of `HBackgroundDevice` and if `device1.setBackgroundConfiguration(...)` and `device2.setBackgroundConfiguration(...)` would not return the same value when equivalent `HBackgroundConfiguration` instances are specified as arguments, then the two screens are not equivalent with respect to underlying resources;

- if *device1* and *device2* are instances of `HVideoDevice` and if (1) *device1.setVideoConfiguration(...)* and *device2.setVideoConfiguration(...)* would not return the same value when equivalent `HVideoConfiguration` instances are specified as arguments, (2) *device1.getVideoSource()* and *device1.getVideoSource()* would not return nominally equivalent video sources, or (3) *device1.getVideoController()* and *device1.getVideoController()* would not return nominally equivalent video controllers, then the two screens are not equivalent with respect to underlying resources;
- if *device1* and *device2* are instances of `HGraphicsDevice` and if *device1.setGraphicsConfiguration(...)* and *device2.setGraphicsConfiguration(...)* would not return the same value when equivalent `HGraphicsConfiguration` instances are specified as arguments, then the two screens are not equivalent with respect to underlying resources;
- if none the above conditions apply, then *device1* and *device2* are deemed equivalent with respect to underlying resources;

Parameters:

device1 - an `HScreenDevice` instance.

device2 - an `HScreenDevice` instance.

Returns:

true if the specified screens are equivalent as described by the above conditions.

Since:

MSM I01

sameResources

```
public boolean sameResources(javax.tv.service.selection.ServiceContext sc1,  
                               javax.tv.service.selection.ServiceContext sc2)
```

Determines if two `ServiceContext` instances represent the same underlying platform resources and underlying resource state, i.e., are equivalent with respect to these underlying resources.

Parameters:

sc1 - a `ServiceContext` instance.

sc2 - a `ServiceContext` instance.

Returns:

true if the specified service contexts are equivalent (i.e., represent the same underlying resources and resource state).

Since:

MSM I01

getInstance

```
public static MultiScreenManager getInstance()
```

Gets the singleton instance of the `MultiScreenManager`.

Returns:

The `MultiScreenManager` instance.

Since:

MSM I01

Package org.ocap.ui.event

Extensions to HAVi User Interface Event classes, including OCAP specific remote control events and multiscreen management events.

Interface Summary

MultiScreenConfigurationListener	This listener is used to provide notifications regarding system and application induced changes to the global state of the <code>MultiScreenManager</code> instance or the state of some display <code>HScreen</code> with respect to the per-platform or some per-display multiscreen configuration, respectively, or to changes to a specific <code>MultiScreenConfiguration</code> instance.
MultiScreenContextListener	This listener is used to provide notifications regarding system and application induced changes to a <code>MultiScreenContext</code> .

Class Summary

MultiScreenConfigurationEvent	A <code>MultiScreenConfigurationEvent</code> is used to report changes to the global state of the <code>MultiScreenManager</code> instance or the state of some display <code>HScreen</code> with respect to the per-platform or some per-display multiscreen configuration, respectively, or to changes to a specific <code>MultiScreenConfiguration</code> instance.
MultiScreenContextEvent	A <code>MultiScreenContextEvent</code> is used to report a change to a <code>MultiScreenContext</code> to interested listeners.
MultiScreenEvent	A <code>MultiScreenEvent</code> is an abstract, base class used to organize event identification codes used by disparate types of events related to multiple screen management functionality.
MultiScreenResourceEvent	A <code>MultiScreenResourceEvent</code> is used to report changes regarding the resource status of multiscreen related resources.

org.ocap.ui.event**Class MultiScreenConfigurationEvent**

```

java.lang.Object
├─ java.util.EventObject
│   └─ org.ocap.ui.event.MultiScreenEvent
│       └─ org.ocap.ui.event.MultiScreenConfigurationEvent

```

All Implemented Interfaces:

```
java.io.Serializable
```

```

public class MultiScreenConfigurationEvent
extends MultiScreenEvent

```

A `MultiScreenConfigurationEvent` is used to report changes to the global state of the `MultiScreenManager` instance or the state of some display `HScreen` with respect to the per-platform or some per-display multiscreen configuration, respectively, or to changes to a specific `MultiScreenConfiguration` instance.

The following types of changes **SHALL** cause the generation of this event:

- The currently active per-platform multiscreen configuration as determined by the `MultiScreenManager` changes from one multiscreen configuration to another multiscreen configuration;
- The currently active per-display multiscreen configuration as determined by some display `HScreen` changes from one multiscreen configuration to another multiscreen configuration;
- The set of screens associated with a `MultiScreenConfiguration` changes (i.e., a screen is added or removed from the multiscreen configuration);

Since:

MSM I01

Field Summary

static int	MULTI_SCREEN_CONFIGURATION_CHANGED The currently active per-platform or some per-display <code>MultiScreenConfiguration</code> as determined by the <code>MultiScreenManager</code> or some display <code>HScreen</code> has changed, in which case the value returned by <code>getSource()</code> SHALL be the affected <code>MultiScreenManager</code> or display <code>HScreen</code> , and the value returned by <code>getRelated()</code> SHALL be the previously active <code>MultiScreenConfiguration</code> .
static int	MULTI_SCREEN_CONFIGURATION_CHANGING A change to the currently active per-platform or some per-display <code>MultiScreenConfiguration</code> as determined by the <code>MultiScreenManager</code> or some display <code>HScreen</code> has been initiated, in which case the value returned by <code>getSource()</code> SHALL be the affected <code>MultiScreenManager</code> or display <code>HScreen</code> , and the value returned by <code>getRelated()</code> SHALL be the subsequently active <code>MultiScreenConfiguration</code> .
static int	MULTI_SCREEN_CONFIGURATION_LAST Last event identifier assigned to <code>MultiScreenConfigurationEvent</code> event identifiers.

Field Summary

static int	<p>MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED</p> <p>The set of screens associated with a MultiScreenConfiguration has changed, with a new screen having been added, in which case the value returned by getSource() SHALL be the affected MultiScreenConfiguration, and the value returned by getRelated() SHALL be the newly added HScreen.</p>
static int	<p>MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED</p> <p>The set of screens associated with a MultiScreenConfiguration has changed, with an existing screen having been removed, in which case the value returned by getSource() SHALL be the affected MultiScreenConfiguration, and the value returned by getRelated() SHALL be the newly removed HScreen.</p>

Fields inherited from class org.ocap.ui.event.MultiScreenEvent

MULTI_SCREEN_CONFIGURATION_FIRST, MULTI_SCREEN_CONTEXT_FIRST

Fields inherited from class java.util.EventObject

source

Constructor Summary

MultiScreenConfigurationEvent(java.lang.Object source, int id, java.lang.Object related)
 Construct an MultiScreenConfigurationEvent.

Method Summary

java.lang.Object	<p>getRelated()</p> <p>Obtain a related object associated with this event.</p>
------------------	---

Methods inherited from class org.ocap.ui.event.MultiScreenEvent

getId

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

MULTI_SCREEN_CONFIGURATION_CHANGING

public static final int **MULTI_SCREEN_CONFIGURATION_CHANGING**

A change to the currently active per-platform or some per-display `MultiScreenConfiguration` as determined by the `MultiScreenManager` or some display `HScreen` has been initiated, in which case the value returned by `getSource()` SHALL be the affected `MultiScreenManager` or display `HScreen`, and the value returned by `getRelated()` SHALL be the subsequently active `MultiScreenConfiguration`.

A `MULTI_SCREEN_CONFIGURATION_CHANGING` event SHALL NOT be dispatched to an application that has not been granted `MonitorAppPermission("multiscreen.configuration")`.

Since:

MSM I01

MULTI_SCREEN_CONFIGURATION_CHANGED

public static final int **MULTI_SCREEN_CONFIGURATION_CHANGED**

The currently active per-platform or some per-display `MultiScreenConfiguration` as determined by the `MultiScreenManager` or some display `HScreen` has changed, in which case the value returned by `getSource()` SHALL be the affected `MultiScreenManager` or display `HScreen`, and the value returned by `getRelated()` SHALL be the previously active `MultiScreenConfiguration`.

Since:

MSM I01

MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED

public static final int **MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED**

The set of screens associated with a `MultiScreenConfiguration` has changed, with a new screen having been added, in which case the value returned by `getSource()` SHALL be the affected `MultiScreenConfiguration`, and the value returned by `getRelated()` SHALL be the newly added `HScreen`.

Except during the interval between the last dispatching of an `MULTI_SCREEN_CONFIGURATION_CHANGING` event and the generation of a corresponding `MULTI_SCREEN_CONFIGURATION_CHANGED` event, a screen SHALL NOT be added to and a `MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED` event SHALL NOT be generated for a multiscreen configuration that is the current per-platform or some current per-display multiscreen configuration.

Since:

MSM I01

MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED

public static final int **MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED**

The set of screens associated with a `MultiScreenConfiguration` has changed, with an existing screen having been removed, in which case the value returned by `getSource()` SHALL be the affected `MultiScreenConfiguration`, and the value returned by `getRelated()` SHALL be the newly removed `HScreen`.

Except during the interval between the last dispatching of an `MULTI_SCREEN_CONFIGURATION_CHANGING` event and the generation of a corresponding

MULTI_SCREEN_CONFIGURATION_CHANGED event, a screen SHALL NOT be removed from and a MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED event SHALL NOT be generated for a multiscreen configuration that is the current per-platform or some current per-display multiscreen configuration.

Since:
MSM I01

MULTI_SCREEN_CONFIGURATION_LAST

```
public static final int MULTI_SCREEN_CONFIGURATION_LAST
    Last event identifier assigned to MultiScreenConfigurationEvent event identifiers.
```

Since:
MSM I01

Constructor Detail

MultiScreenConfigurationEvent

```
public MultiScreenConfigurationEvent(java.lang.Object source,
                                     int id,
                                     java.lang.Object related)
```

Construct an MultiScreenConfigurationEvent.

Parameters:

source - A reference to a MultiScreenManager instance, a display HScreen instance, or a MultiScreenConfiguration instance in accordance with the specific event as specified above.

id - The event identifier of this event, the value of which SHALL be one of the following:

MULTI_SCREEN_CONFIGURATION_CHANGING, MULTI_SCREEN_CONFIGURATION_CHANGED, MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED, or MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED.

related - A reference to a MultiScreenConfiguration instance or an HScreen instance in accordance with the specific event as specified above.

Since:
MSM I01

Method Detail

getRelated

```
public java.lang.Object getRelated()
    Obtain a related object associated with this event.
```

Returns:

The related object instance of this event, the value of which SHALL be one of the following as determined by the specific event type: a reference to a MultiScreenConfiguration instance or a reference to an HScreen instance.

Since:
MSM I01

org.ocap.ui.event

Interface MultiScreenConfigurationListener

All Superinterfaces:

java.util.EventListener

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

This listener is used to provide notifications regarding system and application induced changes to the global state of the MultiScreenManager instance or the state of some display HScreen with respect to the per-platform or some per-display multiscreen configuration, respectively, or to changes to a specific MultiScreenConfiguration instance.

Since:

MSM I01

Method Summary

void	notify (MultiScreenConfigurationEvent evt)
------	---

	When a MultiScreenConfigurationEvent is generated, the implementation SHALL invoke this method on all registered listeners in order to report event information to each listener as required by specific event semantics.
--	---

Method Detail

notify

```
void notify(MultiScreenConfigurationEvent evt)
```

When a MultiScreenConfigurationEvent is generated, the implementation SHALL invoke this method on all registered listeners in order to report event information to each listener as required by specific event semantics.

In case the event is

MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING, this method SHALL NOT be invoked unless the application that registered this listener has been granted MonitorAppPermission("multiscreen.configuration"). Furthermore, an implementation of this method SHOULD severely limit the amount of processing that may occur, since an absolute time limit is placed on the invocation of this method for the set of all applicable listeners.

Parameters:

evt - A MultiScreenConfigurationEvent instance.

Since:

MSM I01

org.ocap.ui.event
Class MultiScreenContextEvent

```
java.lang.Object
├ java.util.EventObject
│   └ org.ocap.ui.event.MultiScreenEvent
│       └ org.ocap.ui.event.MultiScreenContextEvent
```

All Implemented Interfaces:

```
java.io.Serializable
```

```
public class MultiScreenContextEvent
extends MultiScreenEvent
```

A `MultiScreenContextEvent` is used to report a change to a `MultiScreenContext` to interested listeners.

The following types of changes cause the generation of this event:

- Change of associated `ServiceContext`;
- Change of associated display `HScreen`;
- Change of associated display `HScreen` area (extent) assignment;
- Change of associated set of `VideoOutputPorts`;
- Change of audio focus of a display `HScreen`;
- Change of screen visibility;
- Change of screen z-order.
- Change of set of underlying `HScreenDevice` that contribute audio sources to an `HScreen`;
- Change of set of underlying `HScreenDevice` instances, e.g., due to addition or removal of a `HScreenDevice` from an `HScreen`;
- Change of the z-order of the underlying `HScreenDevice` instances of an `HScreen`;

Since:

MSM I01

Field Summary	
static int	<p>MULTI_SCREEN_CONTEXT_AUDIO_FOCUS_CHANGED</p> <p>The audio focus screen of the underlying <code>HScreen</code> of the source <code>MultiScreenContext</code> has changed.</p>
static int	<p>MULTI_SCREEN_CONTEXT_AUDIO_SOURCES_CHANGED</p> <p>The audio sources of the underlying <code>HScreen</code> of the source <code>MultiScreenContext</code> has changed.</p>

Field Summary

static int	MULTI_SCREEN_CONTEXT_DEVICES_CHANGED The set of HScreenDevice instances associated with the underlying HScreen of the source MultiScreenContext has changed.
static int	MULTI_SCREEN_CONTEXT_DEVICES_Z_ORDER_CHANGED The z-order of the set of HScreenDevice instances associated with the underlying HScreen of the source MultiScreenContext has changed.
static int	MULTI_SCREEN_CONTEXT_DISPLAY_AREA_CHANGED The area (extent) of the display HScreen to which the underlying HScreen of the source MultiScreenContext is assigned has changed.
static int	MULTI_SCREEN_CONTEXT_DISPLAY_SCREEN_CHANGED The display HScreen associated with the underlying HScreen of the source code>MultiScreenContext has <changed.
static int	MULTI_SCREEN_CONTEXT_OUTPUT_PORT_CHANGED The set of video output ports associated with underlying HScreen of the source MultiScreenContext has changed.
static int	MULTI_SCREEN_CONTEXT_SERVICE_CONTEXT_CHANGED The ServiceContext associated with the underlying HScreen of the source MultiScreenContext has changed.
static int	MULTI_SCREEN_CONTEXT_VISIBILITY_CHANGED The visibility of the underlying HScreen of the source MultiScreenContext has changed.
static int	MULTI_SCREEN_CONTEXT_Z_ORDER_CHANGED The z-order of the underlying HScreen of the source MultiScreenContext has changed.
static int	MULTI_SCREEN_CONTEXTS_LAST Last event identifier assigned to MultiScreenConfigurationEvent event identifiers.

Fields inherited from class org.ocap.ui.event.MultiScreenEvent

MULTI_SCREEN_CONFIGURATION_FIRST, MULTI_SCREEN_CONTEXT_FIRST

Fields inherited from class java.util.EventObject

source

Constructor Summary

MultiScreenContextEvent(java.lang.Object source, int id)
Construct a MultiScreenContextEvent.

Method Summary

Methods inherited from class org.ocap.ui.event.MultiScreenEvent

getId

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**MULTI_SCREEN_CONTEXT_DEVICES_CHANGED**public static final int **MULTI_SCREEN_CONTEXT_DEVICES_CHANGED**

The set of HScreenDevice instances associated with the underlying HScreen of the source MultiScreenContext has changed.

Since:

MSM I01

MULTI_SCREEN_CONTEXT_DEVICES_Z_ORDER_CHANGEDpublic static final int **MULTI_SCREEN_CONTEXT_DEVICES_Z_ORDER_CHANGED**

The z-order of the set of HScreenDevice instances associated with the underlying HScreen of the source MultiScreenContext has changed.

Since:

MSM I01

MULTI_SCREEN_CONTEXT_SERVICE_CONTEXT_CHANGEDpublic static final int **MULTI_SCREEN_CONTEXT_SERVICE_CONTEXT_CHANGED**

The ServiceContext associated with the underlying HScreen of the source MultiScreenContext has changed.

Since:

MSM I01

MULTI_SCREEN_CONTEXT_DISPLAY_SCREEN_CHANGEDpublic static final int **MULTI_SCREEN_CONTEXT_DISPLAY_SCREEN_CHANGED**

The display HScreen associated with the underlying HScreen of the source code>MultiScreenContext has

Since:

MSM I01

MULTI_SCREEN_CONTEXT_DISPLAY_AREA_CHANGEDpublic static final int **MULTI_SCREEN_CONTEXT_DISPLAY_AREA_CHANGED**

The area (extent) of the display HScreen to which the underlying HScreen of the source MultiScreenContext is assigned has changed.

Since:

MSM I01

MULTI_SCREEN_CONTEXT_OUTPUT_PORT_CHANGED

```
public static final int MULTI_SCREEN_CONTEXT_OUTPUT_PORT_CHANGED
```

The set of video output ports associated with underlying HScreen of the source MultiScreenContext has changed.

Since:
MSM I01

MULTI_SCREEN_CONTEXT_VISIBILITY_CHANGED

```
public static final int MULTI_SCREEN_CONTEXT_VISIBILITY_CHANGED
```

The visibility of the underlying HScreen of the source MultiScreenContext has changed.

Since:
MSM I01

MULTI_SCREEN_CONTEXT_Z_ORDER_CHANGED

```
public static final int MULTI_SCREEN_CONTEXT_Z_ORDER_CHANGED
```

The z-order of the underlying HScreen of the source MultiScreenContext has changed.

Since:
MSM I01

MULTI_SCREEN_CONTEXT_AUDIO_SOURCES_CHANGED

```
public static final int MULTI_SCREEN_CONTEXT_AUDIO_SOURCES_CHANGED
```

The audio sources of the underlying HScreen of the source MultiScreenContext has changed.

Since:
MSM I01

MULTI_SCREEN_CONTEXT_AUDIO_FOCUS_CHANGED

```
public static final int MULTI_SCREEN_CONTEXT_AUDIO_FOCUS_CHANGED
```

The audio focus screen of the underlying HScreen of the source MultiScreenContext has changed. When the audio focus screen of a display HScreen changes, then this event SHALL be generated twice (after completing the change): firstly to the MultiScreenContext of the logical screen which has lost audio focus (if such logical screen existed), and secondly to the MultiScreenContext of the display screen. In both of these cases, the source MultiScreenContext SHALL be the display screen.

Since:
MSM I01

MULTI_SCREEN_CONTEXTS_LAST

```
public static final int MULTI_SCREEN_CONTEXTS_LAST
```

Last event identifier assigned to MultiScreenConfigurationEvent event identifiers.

Since:
MSM I01

Constructor Detail

MultiScreenContextEvent

```
public MultiScreenContextEvent(java.lang.Object source,  
                               int id)
```

Construct a MultiScreenContextEvent.

Parameters:

source - A reference to a MultiScreenContext interface.

id - The event identifier of this event, the value of which SHALL be one of the following:

MULTI_SCREEN_CONTEXT_DEVICES_CHANGED,
MULTI_SCREEN_CONTEXT_DEVICES_Z_ORDER_CHANGED,
MULTI_SCREEN_CONTEXT_SERVICE_CONTEXT_CHANGED,
MULTI_SCREEN_CONTEXT_DISPLAY_SCREEN_CHANGED,
MULTI_SCREEN_CONTEXT_DISPLAY_AREA_CHANGED,
MULTI_SCREEN_CONTEXT_OUTPUT_PORT_CHANGED,
MULTI_SCREEN_CONTEXT_VISIBILITY_CHANGED,
MULTI_SCREEN_CONTEXT_Z_ORDER_CHANGED,
MULTI_SCREEN_CONTEXT_AUDIO_SOURCES_CHANGED, or
MULTI_SCREEN_CONTEXT_AUDIO_FOCUS_CHANGED.

Since:

MSM I01

org.ocap.ui.event

Interface **MultiScreenContextListener**

All Superinterfaces:

java.util.EventListener

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

This listener is used to provide notifications regarding system and application induced changes to a `MultiScreenContext`.

Since:

OCAP 1.0

Method Summary

void	notify (MultiScreenContextEvent evt)
------	---

When an OCAP implementation makes any change to a `MultiScreenContext` that causes generation of a `MultiScreenContextEvent`, then the implementation SHALL invoke this method on all registered listeners in order to report change information to the listener.

Method Detail

notify

```
void notify(MultiScreenContextEvent evt)
```

When an OCAP implementation makes any change to a `MultiScreenContext` that causes generation of a `MultiScreenContextEvent`, then the implementation SHALL invoke this method on all registered listeners in order to report change information to the listener.

If the application that registered this listener has not been granted `MonitorAppPermission("multiscreen.context")` and the source `MultiScreenContext` associated with the specified `MultiScreenContextEvent` is associated with no `ServiceContext` or a `ServiceContext` that is not accessible to that application, then this method SHALL NOT be invoked on this listener; otherwise it SHALL be invoked on this listener.

A `ServiceContext` is accessible to an application if it is returned from the `ServiceContextFactory.getServiceContexts()` method.

Parameters:

evt - A `MultiScreenContextEvent` instance.

Since:

MSM I01

org.ocap.ui.event**Class MultiScreenEvent**

```

java.lang.Object
├─ java.util.EventObject
└─ org.ocap.ui.event.MultiScreenEvent

```

All Implemented Interfaces:

```
java.io.Serializable
```

Direct Known Subclasses:

```
MultiScreenConfigurationEvent, MultiScreenContextEvent
```

```

public abstract class MultiScreenEvent
extends java.util.EventObject

```

A `MultiScreenEvent` is an abstract, base class used to organize event identification codes used by disparate types of events related to multiple screen management functionality.

Since:

```
MSM I01
```

Field Summary

static int	MULTI_SCREEN_CONFIGURATION_FIRST First event identifier assigned to <code>MultiScreenConfigurationEvent</code> event identifiers.
static int	MULTI_SCREEN_CONTEXT_FIRST First event identifier assigned to <code>MultiScreenContextEvent</code> event identifiers.

Fields inherited from class java.util.EventObject

```
source
```

Constructor Summary

protected	MultiScreenEvent (java.lang.Object source, int id) Protected constructor for an <code>MultiScreenEvent</code> .
-----------	---

Method Summary

int	getId() Obtain the event identifier associated with this event.
-----	---

Methods inherited from class java.util.EventObject

```
getSource, toString
```

Methods inherited from class java.lang.Object

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

Field Detail

MULTI_SCREEN_CONFIGURATION_FIRST

```
public static final int MULTI_SCREEN_CONFIGURATION_FIRST
```

First event identifier assigned to MultiScreenConfigurationEvent event identifiers.

Since:

MSM I01

MULTI_SCREEN_CONTEXT_FIRST

```
public static final int MULTI_SCREEN_CONTEXT_FIRST
```

First event identifier assigned to MultiScreenContextEvent event identifiers.

Since:

MSM I01

Constructor Detail

MultiScreenEvent

```
protected MultiScreenEvent(java.lang.Object source,  
                             int id)
```

Protected constructor for an MultiScreenEvent.

Parameters:

source - A reference to an event source as defined by a concrete subclass of this class.

id - The event identifier of this event.

Since:

MSM I01

Method Detail

getId

```
public int getId()
```

Obtain the event identifier associated with this event.

Returns:

The event identifier of this event, for which see the sub-classes of this class:

MultiScreenConfigurationEvent and MultiScreenContextEvent.

Since:

MSM I01

org.ocap.ui.event
Class MultiScreenResourceEvent

```
java.lang.Object
├─ java.util.EventObject
│   └─ org.davic.resources.ResourceStatusEvent
│       └─ org.ocap.ui.event.MultiScreenResourceEvent
```

All Implemented Interfaces:

```
java.io.Serializable
```

```
public class MultiScreenResourceEvent
extends org.davic.resources.ResourceStatusEvent
```

A MultiScreenResourceEvent is used to report changes regarding the resource status of multiscreen related resources.

Since:

MSM I01

Field Summary	
static int	MULTI_SCREEN_RESOURCE_SCREEN_RELEASED The reservation on a screen has just been released, indicating that the screen (or its constituent screen devices) MAY now be reserved (i.e., they are now unreserved).
static int	MULTI_SCREEN_RESOURCE_SCREEN_RESERVED The reservation on a screen has just been granted to an application, indicating that the screen (including its constituent screen devices) is no longer unreserved.

Fields inherited from class java.util.EventObject
source

Constructor Summary
MultiScreenResourceEvent (java.lang.Object source, int id) Constructor for an MultiScreenResourceEvent.

Method Summary	
int	getId() Obtain the resource event identifier associated with this event.
java.lang.Object	getSource() Obtain the source object that generated this event.

Methods inherited from class java.util.EventObject
toString

Methods inherited from class java.lang.Object

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

Field Detail**MULTI_SCREEN_RESOURCE_SCREEN_RELEASED**

```
public static final int MULTI_SCREEN_RESOURCE_SCREEN_RELEASED
```

The reservation on a screen has just been released, indicating that the screen (or its constituent screen devices) MAY now be reserved (i.e., they are now unreserved).

Since:

MSM I01

MULTI_SCREEN_RESOURCE_SCREEN_RESERVED

```
public static final int MULTI_SCREEN_RESOURCE_SCREEN_RESERVED
```

The reservation on a screen has just been granted to an application, indicating that the screen (including its constituent screen devices) is no longer unreserved.

Since:

MSM I01

Constructor Detail**MultiScreenResourceEvent**

```
public MultiScreenResourceEvent(java.lang.Object source,  
                                int id)
```

Constructor for an MultiScreenResourceEvent.

Parameters:

source - A reference to an HScreen instance whose resource status has changed.

id - The event identifier of this event.

Since:

MSM I01

Method Detail**getSource**

```
public java.lang.Object getSource()
```

Obtain the source object that generated this event.

Overrides:

getSource in class org.davic.resources.ResourceStatusEvent

Returns:

A reference to an HScreen instance, or a subclass thereof.

Since:

MSM I01

getId

```
public int getId()
```

Obtain the resource event identifier associated with this event.

Returns:

The event identifier of this event, where the identifier is one of the following: {
MULTI_SCREEN_RESOURCE_SCREEN_RELEASED,
MULTI_SCREEN_RESOURCE_SCREEN_RESERVED }.

Since:

MSM I01

Annex B Multiscreen Reconfiguration Constraints

The assertions defined by the method `verifyConstraints()` below apply during multiscreen configuration change processing:

```

/**
 * Reconfiguration Constraints
 *
 * Assumptions:
 *
 * 1. platform implements MSM as defined;
 * 2. more than one distinct multiscreen configuration exists and is
 *    accessible;
 * 3. caller has MonitorAppPermission("multiscreen.configuration");
 * 4. no MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING or
 *    MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED event
 *    is generated during the execution of this method other than as a direct
 *    result of the invocation of setMultiScreenConfiguration() by this method;
 * 5. no MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_SCREEN_ADDED
 *    or MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_SCREEN_REMOVED
 *    event is generated during the execution of this method;
 * 6. application signals allow_default_device_reconfig as '1' in multiple
 *    screen usage descriptor;
 * 7. the platform preserves key default screen device configuration parameters,
 *    specifically screen area, pixel resolution, and pixel aspect ratio across
 *    configuration change; i.e., it does not take advantage of fact that
 *    application has signaled allow_default_device_reconfig as '1' in order to
 *    reconfigure default screen device parameters (note that this verification
 *    process could be expanded to cover case where platform does not preserve
 *    these parameters, i.e., an appropriate HScreenConfigurationEvent or
 *    MultiScreenContextEvent could be listened for and used to note that
 *    platform does not preserve these parameters);
 */
static void verifyConstraints()
{
    MultiScreenManager      MSM                = MultiScreenManager.getInstance();

    // PRECONDITIONS

    // verify invariants
    verifyInvariantConstraints();

    // record default screen
    HScreen                 S1                = MSM.getDefaultScreen();

    // record default background screen device and key configuration parameters
    HBackgroundDevice       B1                = S1.getDefaultHBackgroundDevice();
    HScreenConfiguration    BC1              = null;
    HScreenRectangle        B1SR             = null;
    Dimension                B1PR            = null;
    Dimension                B1PAR           = null;
    if ( B1 != null ) {
        BC1                  = B1.getCurrentConfiguration();
        B1SR                 = BC1.getScreenArea();
        B1PR                 = BC1.getPixelResolution();
        B1PAR                = BC1.getPixelAspectRatio();
    }
}

```

```

// record default video screen device and key configuration parameters
HVideoDevice          V1              = S1.getDefaultHVideoDevice();
HScreenConfiguration V1C1            = null;
HScreenRectangle     V1SR            = null;
Dimension             V1PR            = null;
Dimension             V1PAR           = null;
if ( V1 != null ) {
    V1C1    = V1.getCurrentConfiguration();
    V1SR    = V1C1.getScreenArea();
    V1PR    = V1C1.getPixelResolution();
    V1PAR   = V1C1.getPixelAspectRatio();
}

// record default graphics screen device and key configuration parameters
HGraphicsDevice      G1              = S1.getDefaultHGraphicsDevice();
HScreenConfiguration G1C1            = null;
HScreenRectangle     G1SR            = null;
Dimension             G1PR            = null;
Dimension             G1PAR           = null;
if ( G1 != null ) {
    G1C1    = G1.getCurrentConfiguration();
    G1SR    = G1C1.getScreenArea();
    G1PR    = G1C1.getPixelResolution();
    G1PAR   = G1C1.getPixelAspectRatio();
}

// record non-default screens
HScreen[]             SX1ND           = getNonDefaultScreens();

// record non-default screen devices
HScreenDevice[]      DX1ND           = getNonDefaultScreenDevices();

// find different configuration from current configuration
MultiScreenConfiguration[] MSCX1     = MSM.getMultiScreenConfigurations();
MultiScreenConfiguration MSC1        = MSM.getMultiScreenConfiguration();
MultiScreenConfiguration MSC2        = null;
for ( int i = 0; ( MSC2 == null ) && ( i < MSCX1.length ); i++ ) {
    if ( MSCX1[i] != MSC1 )
        MSC2 = MSCX1[i];
}
ASSERT ( MSC2 != null );
ASSERT ( MSC2 != MSC1 );

// CHANGE CONFIGURATION
try {
    MSM.setMultiScreenConfiguration ( MSC2, null );
    waitForMultiScreenConfigurationChangingEvent();
    waitForMultiScreenConfigurationChangedEvent();
} catch ( SecurityException x ) {
    ASSERT ( false );
}

// POSTCONDITIONS

// verify invariants
verifyInvariantConstraints();

// 1. the current multiscreen configuration is the new configuration

```

```

ASSERT ( MSM.getMultiScreenConfiguration() == MSC2 );

// 2. the new default screen must be same instance as the old default screen
HScreen          S2                = MSM.getDefaultScreen();
ASSERT ( S2 == S1 );

// 3. if it exists, the new default background screen device
// must be the same instance as the old default background
// screen device if it exists unless the application signals
// allow_default_device_reconfig as '1' in which case it is
// permitted that no default background device is available
// after reconfiguration, in which case the former default
// background device must be reset to the empty background
// screen device state
HBackgroundDevice B2                = S2.getDefaultHBackgroundDevice();
if ( B1 != null ) {
    if ( B2 == null )
        ASSERT ( MSM.isEmptyScreenDevice ( B1 ) );
    else
        ASSERT ( B2 == B1 );
}

// 4. if it exists, the new default background screen device
// must have same screen area, pixel resolution, and pixel
// aspect ratio as it did with previous default background
// screen device
if ( B1 != null ) {
    if ( B2 != null ) {
        HScreenConfiguration BC2                = B1.getCurrentConfiguration();
        HScreenRectangle B2SR                = BC2.getScreenArea();
        ASSERT ( B1SR != null );
        ASSERT ( B2SR != null );
        ASSERT ( B2SR.x == B1SR.x );
        ASSERT ( B2SR.y == B1SR.y );
        ASSERT ( B2SR.width == B1SR.width );
        ASSERT ( B2SR.height == B1SR.height );
        Dimension B2PR                = BC2.getPixelResolution();
        ASSERT ( B1PR != null );
        ASSERT ( B2PR != null );
        ASSERT ( B2PR.width == B1PR.width );
        ASSERT ( B2PR.height == B1PR.height );
        Dimension B2PAR                = BC2.getPixelAspectRatio();
        ASSERT ( B1PAR != null );
        ASSERT ( B2PAR != null );
        ASSERT ( B2PAR.width == B1PAR.width );
        ASSERT ( B2PAR.height == B1PAR.height );
    }
}

// 5. if it exists, the new default video screen device
// must be the same instance as the old default video
// screen device if it exists unless the application signals
// allow_default_device_reconfig as '1' in which case it is
// permitted that no default video device is available
// after reconfiguration, in which case the former default
// video device must be reset to the empty video
// screen device state
HVideoDevice      V2                = S2.getDefaultHVideoDevice();
if ( V1 != null ) {

```

```

    if ( V2 == null )
        ASSERT ( MSM.isEmptyScreenDevice ( V1 ) );
    else
        ASSERT ( V2 == V1 );
}

// 6. if it exists, the new default video screen device
// must have same screen area, pixel resolution, and pixel
// aspect ratio as it did with previous default video
// screen device
if ( V1 != null ) {
    if ( V2 != null ) {
        HScreenConfiguration    VC2                = V1.getCurrentConfiguration();
        HScreenRectangle        V2SR                = VC2.getScreenArea();
        ASSERT ( V1SR != null );
        ASSERT ( V2SR != null );
        ASSERT ( V2SR.x == V1SR.x );
        ASSERT ( V2SR.y == V1SR.y );
        ASSERT ( V2SR.width == V1SR.width );
        ASSERT ( V2SR.height == V1SR.height );
        Dimension                V2PR                = VC2.getPixelResolution();
        ASSERT ( V1PR != null );
        ASSERT ( V2PR != null );
        ASSERT ( V2PR.width == V1PR.width );
        ASSERT ( V2PR.height == V1PR.height );
        Dimension                V2PAR               = VC2.getPixelAspectRatio();
        ASSERT ( V1PAR != null );
        ASSERT ( V2PAR != null );
        ASSERT ( V2PAR.width == V1PAR.width );
        ASSERT ( V2PAR.height == V1PAR.height );
    }
}

// 7. if it exists, the new default graphics screen device
// must be the same instance as the old default graphics
// screen device if it exists unless the application signals
// allow_default_device_reconfig as '1' in which case it is
// permitted that no default graphics device is available
// after reconfiguration, in which case the former default
// graphics device must be reset to the empty graphics
// screen device state
HGraphicsDevice                G2                = S2.getDefaultHGraphicsDevice();
if ( G1 != null ) {
    if ( G2 == null )
        ASSERT ( MSM.isEmptyScreenDevice ( G1 ) );
    else
        ASSERT ( G2 == G1 );
}

// 8. if it exists, the new default graphics screen device
// must have same screen area, pixel resolution, and pixel
// aspect ratio as it did with previous default graphics
// screen device
if ( G1 != null ) {
    if ( G2 != null ) {
        HScreenConfiguration    GC2                = G1.getCurrentConfiguration();
        HScreenRectangle        G2SR                = GC2.getScreenArea();
        ASSERT ( G1SR != null );
        ASSERT ( G2SR != null );
    }
}

```

```

    ASSERT ( G2SR.x == G1SR.x );
    ASSERT ( G2SR.y == G1SR.y );
    ASSERT ( G2SR.width == G1SR.width );
    ASSERT ( G2SR.height == G1SR.height );
    Dimension          G2PR          = GC2.getPixelResolution();
    ASSERT ( G1PR != null );
    ASSERT ( G2PR != null );
    ASSERT ( G2PR.width == G1PR.width );
    ASSERT ( G2PR.height == G1PR.height );
    Dimension          G2PAR         = GC2.getPixelAspectRatio();
    ASSERT ( G1PAR != null );
    ASSERT ( G2PAR != null );
    ASSERT ( G2PAR.width == G1PAR.width );
    ASSERT ( G2PAR.height == G1PAR.height );
}
}

// 9. for every non-default screen obtained prior to
// reconfiguration, if no longer a non-default screen, then it
// must be equivalent to an empty screen; otherwise, it must not
// be equivalent to an empty screen
HScreen[]          SX2ND          = getNonDefaultScreens();
for ( int i = 0, j; i < SX1ND.length; i++ ) {
    HScreen          S            = SX1ND[i];
    for ( j = 0; j < SX2ND.length; i++ ) {
        if ( S == SX2ND[j] )
            break;
    }
    if ( j == SX2ND.length )
        ASSERT ( MSM.isEmptyScreen ( S ) );
    else
        ASSERT ( ! MSM.isEmptyScreen ( S ) );
}

// 10. for every non-default screen device obtained prior to
// reconfiguration, if no longer a non-default screen device,
// then it must be equivalent to an empty screen device;
// otherwise, it must not be equivalent to an empty screen
// device
HScreenDevice[]   DX2ND          = getNonDefaultScreenDevices();
for ( int i = 0, j; i < DX1ND.length; i++ ) {
    HScreenDevice   D            = DX1ND[i];
    for ( j = 0; j < DX2ND.length; i++ ) {
        if ( D == DX2ND[j] )
            break;
    }
    if ( j == DX2ND.length )
        ASSERT ( MSM.isEmptyScreenDevice ( D ) );
    else
        ASSERT ( ! MSM.isEmptyScreenDevice ( D ) );
}
}

static private void verifyInvariantConstraints()
{
    int found;

    // 1. there must be a multiscreen manager
    MultiScreenManager MSM          = MultiScreenManager.getInstance();

```

```

ASSERT ( MSM != null );

// 2. there must be a current multiscreen configuration
MultiScreenConfiguration MSC = MSM.getMultiScreenConfiguration();
ASSERT ( MSC != null );

// 3. there must be a non-empty set of accessible multiscreen
// configurations
MultiScreenConfiguration[] MSCX = MSM.getMultiScreenConfigurations();
ASSERT ( MSCX != null );
ASSERT ( MSCX.length > 0 );

// 4. the current multiscreen configuration must be an
// accessible configuration
found = 0;
for ( int i = 0; i < MSCX.length; i++ ) {
    ASSERT ( MSCX[i] != null );
    if ( MSCX[i] == MSC ) { // instance identity required
        found++;
    }
}
ASSERT ( found == 1 );

// 5. there must be a non-empty set of screens in current
// multiscreen configuration
HScreen[] MSCSX = MSC.getScreens();
ASSERT ( MSCSX != null );
ASSERT ( MSCSX.length > 0 );

// 6. the screens in the current multiscreen configuration must
// not be empty
for ( int i = 0; i < MSCSX.length; i++ ) {
    ASSERT ( ! MSM.isEmptyScreen ( MSCSX[i] ) );
}

// 7. any two distinct screen entries in the current multiscreen
// configuration must not represent the same resources
for ( int i = 0; i < MSCSX.length; i++ ) {
    for ( int j = 0; j < MSCSX.length; j++ ) {
        if ( i != j ) {
            ASSERT ( MSCSX[i] != null );
            ASSERT ( MSCSX[j] != null );
            ASSERT ( ! MSM.sameResources ( MSCSX[i], MSCSX[j] ) );
        }
    }
}

// 8. there must be a current default screen
HScreen MSMS0 = MSM.getDefaultScreen();
ASSERT ( MSMS0 != null );

// 9. the current default screen must not be equivalent to the
// empty screen
ASSERT ( ! MSM.isEmptyScreen ( MSMS0 ) );

// 10. exactly one screen entry in the current multiscreen
// configuration must represent the same resources as the default screen
found = 0;
for ( int i = 0; i < MSCSX.length; i++ ) {

```

```

        if ( MSM.sameResources ( MSCSX[i], MSMS0 ) )
            found++;
    }
    ASSERT ( found == 1 );

    // 11. there must be a non-empty set of accessible screens
    HScreen[]          MSMSX          = MSM.getScreens();
    ASSERT ( MSMSX != null );
    ASSERT ( MSMSX.length > 0 );

    // 12. the current default screen must be a distinct member of
    // the set of accessible screens

    found = 0;
    for ( int i = 0; i < MSMSX.length; i++ ) {
        ASSERT ( MSMSX[i] != null );
        if ( MSMSX[i] == MSMS0 ) { // instance identity required
            found++;
        }
    }
    ASSERT ( found == 1 );

    // 13. any background screen device of the current default
    // screen must not be equivalent to the empty background screen device
    HBackgroundDevice[] BX          = MSMS0.getHBackgroundDevices();
    if ( BX != null ) {
        for ( int i = 0; i < BX.length; i++ ) {
            ASSERT ( BX[i] != null );
            ASSERT ( ! MSM.isEmptyScreenDevice ( BX[i] ) );
        }
    }

    // 14. any video screen device of the current default screen
    // must not be equivalent to the empty video screen device
    HVideoDevice[]      VX          = MSMS0.getHVideoDevices();
    if ( VX != null ) {
        for ( int i = 0; i < VX.length; i++ ) {
            ASSERT ( VX[i] != null );
            ASSERT ( ! MSM.isEmptyScreenDevice ( VX[i] ) );
        }
    }

    // 15. any graphics screen device of the current default screen
    // must not be equivalent to the empty graphics screen device
    HGraphicsDevice[]   GX          = MSMS0.getHGraphicsDevices();
    if ( GX != null ) {
        for ( int i = 0; i < GX.length; i++ ) {
            ASSERT ( GX[i] != null );
            ASSERT ( ! MSM.isEmptyScreenDevice ( GX[i] ) );
        }
    }
}

static private HScreen[] getNonDefaultScreens()
{
    MultiScreenManager MSM          = MultiScreenManager.getInstance();
    HScreen[]          SX1          = MSM.getScreens();
    HScreen            S1           = MSM.getDefaultScreen();
}

```

```

int                NSX1ND                = 0;
for ( int i = 0; i < SX1.length; i++ ) {
    if ( ! MSM.sameResources ( SX1[i], S1 ) )
        NSX1ND++;
}
HScreen[]          SX1ND                = new HScreen[NSX1ND];
for ( int i = 0, n = 0; i < SX1.length; i++ ) {
    if ( ! MSM.sameResources ( SX1[i], S1 ) )
        SX1ND[n++] = SX1[i];
}
return SX1ND;
}

static private HScreenDevice[] getNonDefaultScreenDevices()
{
    MultiScreenManager    MSM                = MultiScreenManager.getInstance();
    HScreen[]             SX1                = MSM.getScreens();
    int                   NDX1ND            = 0;
    for ( int i = 0; i < SX1.length; i++ ) {
        HScreenDevice[]   BX1                = SX1[i].getHBackgroundDevices();
        HScreenDevice     B1                = SX1[i].getDefaultHBackgroundDevice();
        for ( int j = 0; i < BX1.length; j++ ) {
            if ( ! MSM.sameResources ( BX1[i], B1 ) )
                NDX1ND++;
        }
        HScreenDevice[]   VX1                = SX1[i].getHVideoDevices();
        HScreenDevice     V1                = SX1[i].getDefaultHVideoDevice();
        for ( int j = 0; i < VX1.length; j++ ) {
            if ( ! MSM.sameResources ( VX1[i], V1 ) )
                NDX1ND++;
        }
        HScreenDevice[]   GX1                = SX1[i].getHGraphicsDevices();
        HScreenDevice     G1                = SX1[i].getDefaultHGraphicsDevice();
        for ( int j = 0; i < GX1.length; j++ ) {
            if ( ! MSM.sameResources ( GX1[i], G1 ) )
                NDX1ND++;
        }
    }
    HScreenDevice[]       DX1ND            = new HScreenDevice[NDX1ND];
    for ( int i = 0, n = 0; i < SX1.length; i++ ) {
        HScreenDevice[]   BX1                = SX1[i].getHBackgroundDevices();
        HScreenDevice     B1                = SX1[i].getDefaultHBackgroundDevice();
        for ( int j = 0; i < BX1.length; j++ ) {
            if ( ! MSM.sameResources ( BX1[i], B1 ) )
                DX1ND[n++] = BX1[i];
        }
        HScreenDevice[]   VX1                = SX1[i].getHVideoDevices();
        HScreenDevice     V1                = SX1[i].getDefaultHVideoDevice();
        for ( int j = 0; i < VX1.length; j++ ) {
            if ( ! MSM.sameResources ( VX1[i], V1 ) )
                DX1ND[n++] = VX1[i];
        }
        HScreenDevice[]   GX1                = SX1[i].getHGraphicsDevices();
        HScreenDevice     G1                = SX1[i].getDefaultHGraphicsDevice();
        for ( int j = 0; i < GX1.length; j++ ) {
            if ( ! MSM.sameResources ( GX1[i], G1 ) )
                DX1ND[n++] = GX1[i];
        }
    }
}

```

```
    return DX1ND;
}

static private void waitForMultiScreenConfigurationChangingEvent()
{
    // wait for MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGING
}

static private void waitForMultiScreenConfigurationChangedEvent()
{
    // wait for MultiScreenConfigurationEvent.MULTI_SCREEN_CONFIGURATION_CHANGED
}

static private void ASSERT ( boolean condition )
{
    // check and report assertion failure
}
```
