# Introduction

This document provides a high level overview of the test xlets that are executed against each RI release. The xlets fall into two groups, those that are run via autoXlet and those which are run manually. Instructions for running each groups of tests, as well as descriptions of the functionality tested are provided in the following sections.

The test xlets are located in *$OCAPROOT/apps/qa/org/cablelabs/xlet/*. All of the test applications listed here have a corresponding readme.txt file. The readme.txt files are fairly up-to-date. However, if there is a discrepancy, please use the information provided in this document as it is the most current.

# Manual Xlet Tests

Below is the set of test xlets which are run manually. Test descriptions and setup instructions are provided in the following sections.

* AppVersionTest
* EAS
* GraphicsDevTest
* PermissionTest – Unsigned
* PorterDuff
* PropertiesTest - Unsigned
* TimeTest
* TuneTest

## AppVersionTest

This application displays the app version that is started.

### Test configuration

In $OCAPROOT/bin/$OCAPTC/env:

* Remove hostapp.properties
* Set final.properties to include:

*OCAP.xait.ignore=false*

*OCAP.appstorage.storeLocalApps=true*

*OCAP.mgrmgr.manager.Signalling= org.cablelabs.impl.manager.signalling.TestSignallingMgr*

### Test Procedure

1. Copy $OCAPROOT/apps/qa/org/cablelabs/xlet/AppVersionTest/xait.properties\_1 to $OCAPROOT/bin/$OCAPTC/env/xait.properties.
2. From $PLATFORMROOT start RI by entering “./runRI.sh” and see Version 1 of the app is started.
3. Copy xait.properties\_2 to $OCAPROOT/bin/$OCAPTC/env/xait.properties. This tests the scenario of an "emergency" replace of the app, where the previous version (1) is signaled to be destroyed and a new version (2) is to start.
4. Verify Version 2 is started; Press VK\_INFO and verify both "hasNewVers" and "isNewVersionSignaled" should be set to false.
5. Copy xait.properties\_3 to $OCAPROOT/bin/$OCAPTC/env/xait.properties. This tests the scenario where two versions of the same app are signaled to autostart in the same service. The version with the higher launch\_order (version 2) should be the version actually shown on the UI as started.
6. Verify Version 2 is started; press VK\_INFO and confirm both "hasNewVers" and "isNewVersionSignaled" should be set to true because Version 3 has been signaled and stored.
7. Copy xait.properties\_4 to $OCAPROOT/bin/$OCAPTC/env/xait.properties. This tests the scenario where two versions of the same app are signaled to autostart in different service. Again, the version with the higher launch order (version 3) is the one that gets started and shown on the UI.
8. Verify Version 3 is started; Press VK\_INFO and verify both "hasNewVers" and "isNewVersionSignaled" should be set to true because Version 4 has been signaled and stored.
9. Copy xait.properties\_5 to $OCAPROOT/bin/$OCAPTC/env/xait.properties. This tests another flavor of the "emergency" replace scenario with the difference being that the new version to be autostarted is in a different service.
10. Verify Version 4 is started. Press VK\_INFO and verify both "hasNewVers" and "isNewVersionSignaled" should be set to false since no other versions of the app is available.

## EAS Test

Basic EAS testing can be done with the use of the RIExerciser xlet.

A number of EAS test MPEG sections streams (.bin files) for the RI PC Platform is included in $RICOMMONROOT/resources/fdcdata/eas-test-files. Each .bin file constitutes a separate alert message:

* Eas\_form1.bin - Snapshot of the original Eas-form1.bin contents.
* EAS\_event\_ID\_2.bin - Simple text-only alert with a finite 5 second presentation time.
* EAS\_event\_ID\_3.bin - Simple details channel alert with a finite 10 second presentation time.
* EAS\_event\_ID\_4.bin - Simple text+audio alert with a finite 15 second presentation time.
* EAS\_event\_ID\_5.bin - Simple text-only alert with an indefinite display time.
* EAS\_event\_ID\_6.bin - Simple details channel alert with an indefinite display time.
* EAS\_event\_ID\_7.bin - Simple text+audio alert with an indefinite display time

### Test Configuration

In $OCAPROOT/bin/$OCAPTC/env set final.properties to include:

OCAP.xait.ignore=true

To enable EAS, edit $RICOMMONROOT/common/resources/fdc-files.txt to include Eas-form1.bin. The EAS alert can be changed by replacing Eas-form1.bin with one of the other EAS\_event\_ID\_\*.bin files. For example:

*> cp $RICOMMONROOT/resources/fdcdata/eas-test-files/EAS\_event\_ID\_7.bin $RICOMMONROOT/resources/fdcdata/Eas-form1.bin*

### Test Procedure

1. Enable EAS by adding the Eas-form1.bin entry to the fdc-files.txt file. fdc-files.txt should contain:

*Ate-SI-data.bin*

*Ate-XAIT.bin*

*Eas-form1.bin*

1. Copy EAS\_event\_ID\_5.bin to Eas-form1.bin:

*cp $RICOMMONROOT/resources/fdcdata/eas-test-files/EAS\_event\_ID\_5.bin $RICOMMONROOT/resources/fdcdata/Eas-form1.bin*

1. From $PLATFORMROOT start RIExerciser by entering:

*./runRI.sh –setup –xlet RiExerciser*

1. Once RIExerciser has started, click on the POWER button to change the power mode to full power.
2. Verify EAS alert is displayed and is not obstructed by the video being presented via RiExerciser.
3. Click on the POWER button again to change the power mode back to low.
4. Verify EAS alert is no longer displayed.
5. Change channel and change the power mode to high again.
6. Verify EAS alert is once again displayed and not obstructed by the video being presented via RIExerciser.
7. Do a few more channel changes. Verify the channel changes are successful and the EAS alert continues to be displayed unobstructed in the new channel
8. Copy EAS\_event\_ID\_6.bin to Eas-form1.bin:

*cp $RICOMMONROOT/resources/fdcdata/eas-test-files/EAS\_event\_ID\_6.bin $RICOMMONROOT/resources/fdcdata/Eas-form1.bin*

1. Start RI/RIExerciser as described above.
2. Once RIExerciser has started, click on the POWER button to change the power mode to full power.
3. Verify EAS alert forces tune to details channel.
4. Click on the POWER button again to change the power mode back to low.
5. Verify details channel is no longer displayed.
6. Change channel and change the power mode to high again.
7. Verify EAS alert forces tune to details channel. Do a few more channel changes. Verify the channel change is prevented.
8. Change the EAS alert message by copying a different .bin to **Eas-form1.bin** and verify the alert message being display does change accordingly.

## GraphicsDev Test

This test checks that a call to HGraphicsDevice.setGraphicsConfiguration works properly. It changes the graphics device configuration (and the coherent config) through a call to HGraphicsDevice.setGraphicsConfiguration.

There are 4 available graphics configurations; press the '0", "1", "2" and "3" keys to select one of them. The log will contain messages detailing what the current graphics resolution is. The selections correspond to:

"0": 640x480 1:1 (SD)

"1": 960x540 3:4 (SD)

"2": 640x480 4:3 (HD)

"3": 960x540 1:1 (HD)

### Test Configuration

In $OCAPROOT/bin/$OCAPTC/env:

* Set final.properties to include:

*OCAP.xait.ignore=true*

* Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/GraphicsDevTest /hostapp.properties.graphicsdevtest* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*

2.3.2 *Test Procedure*

1. From $PLATFORMROOT start the RI by entering ./*runRI.sh.*
2. Press “0”. Verify that the green HScene occupies the entire RI screen.
3. Press “1”. Verify that the green HScene does not fill the entire screen and is flush with the upper left corner of the RI screen.
4. Press “2”. Verify that the green HScene occupies a horizontal band across the RI screen.
5. Press “3”. Verify that the green HScene occupies a rectangle flush with the left side of the RI screen.

## Permissions Tests

Six applications are available to check that the correct permissions are granted or denied based on the application's AppId and associated Permission Request File (PRF).

Note: The tests do not contain a fully featured PRF parser. The current implementation only scans the file for certain tokens and then tests that the requested permission is correctly denied or granted based on the value given in the permission entry. So comments in a PRF have no effect and the XML syntax in the file is not validated.

The test application uses the SecurityManager.checPermission(Permission p) method to determine if a permission has been granted or denied. It assumes that if a SecurityException is thrown, then that permission was denied; otherwise, the permission was granted.

Five of the six Permissions tests are run through the autoXlet method. Only the Unsigned ApplicationPermissionsTest is run manually. A description of all six permissions test is included below as well as instructions for how to run the UnsignedApplicationPermissions Test manually.

**Unsigned Application Permission Test**

This is an application with AppId below 0x4000 (i.e.: 3001). The test verifies it has been granted every permission that any application should have regardless of its AppId. The test also confirms that it does not have any permission that should only be granted to a signed application or one that should only be granted via a PRF file.

**Signed with PRF Application Permission Test**

This is an application with AppId 4001 (i.e. signed app). The application also has an associated PRF file that grants it various permissions.

The application verifies that in addition to the same permissions granted to the unsigned application, it also has the following permissions granted because it is a signed app:

* java.io.FilePermission /oc/- read
* javax.tv.service.selection.ServiceContextPermission \* own
* javax.tv.service.selection.SelectPermission \* own
* java.util.PropertyPermission dvb.persistent.root read

Moreover, the test also checks all the permissions granted via its PRF definition are available to it. Please see *$OCAPROOT/bin/$OCAPTC/env/qa/xlet/org/cablelabs/xlet/PermissionTest/ocap.SignedPermissionTestXlet.perm* for the specific PRF definition.

Furthermore, the xlet confirms that all MonitorApplication level permission denied.

**Signed withOUT PRF Application Permission Test**

This is another signed application with AppId 4006. This test verifies it has the same permissions as the **signed with PRF application** (4001) minus all the permissions only grantable via the PRF file.

**DuallySigned With PRF Application Permission Test**

This is a MonitorApplication level xlet (AppID 6001) with an associated PRF file that grants it various permissions.

The application verifies that in addition to the same permissions granted to the signed application, it was also granted MonitorApplication level permission requested through its PRF file. Please see *$OCAPROOT/bin/$OCAPTC/env/qa/xlet/org/cablelabs/xlet/PermissionTest/ocap.DuallySignedPermissionsTestXlett.perm* for the specific PRF definition.

**DuallySigned WithOUT PRF Application Permission Test**

This is another MonitorApplication level xlet with AppId 6202. This test verifies it has the same permissions as the **duallySigned with PRF application** (6001) minus any of the permissions that can only be granted via a PRF file.

**SuperHost With PRF Application Permission Test**

This is a SuperHost application with AppId 7002 with an associated PRF that grants it additional permission.

The application verifies that in addition to the same permissions granted to the signed application, it was also granted MonitorApplication level permission requested through its PRF file. Please see *$OCAPROOT/bin/$OCAPTC/env/qa/xlet/org/cablelabs/xlet/ PermissionTest/ocap.HostAppPermissionsTestXlett.perm* for the specific PRF definition.

### Test Configuration

In $OCAPROOT/bin/$OCAPTC/env:

* Set final.properties to include:

*OCAP.xait.ignore=true*

* Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/PermissionTest/ hostapp.properties.perm* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*

### Test Procedure

1. Edit hostapp.properties so the application\_control\_code for the application to be run is set to AUTOSTART (instead of PRESENT). In this case the PermissionTest – Unsigned App will be set to AUTOSTART.
2. From $PLATFORMROOT start the RI by entering ./*runRI.sh.*
3. If the test application is granted or denied any unexpected permissions, the unexpected permission will be printed to the screen over a red background. Pass result will be shown over a green background.

## PorterDuff Test

This xlet tests the the PorterDuff rules for graphics composition. There are 8 available PorterDuff rules; press the "1" - "8" keys to select one of them. The selections correspond to:

1: DVBAlphaComposite.Clear

2. DVBAlphaComposite.Src

3. DVBAlphaComposite.SrcOver

4. DVBAlphaComposite.DstOver

5. DVBAlphaComposite.SrcIn

6. DVBAlphaComposite.DstIn

7. DVBAlphaComposite.SrcOut

8. DVBAlphaComposite.DstOut

The xlet displays 2 overlapping rectangles, one blue and one red, each with an alpha of 0.5.

### Test Configuration

In $OCAPROOT/bin/$OCAPTC/env:

* Set final.properties to include:

*OCAP.xait.ignore=true*

* Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/ PorterDuffTest / hostapp.properties.porterd* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*

### Test Procedure

1. From $PLATFORMROOT start the RI by entering ./*runRI.sh****.***
2. Press “1”. Verify that both rectangles are black (alpha=1) and the background is green.
3. Press “2”. Verify that the red rectangle (alpha=1) is overlayed on the blue rectangle (alpha=1) and the background is green.
4. Press “3”. Verify that the red rectangle (alpha<1) is overlayed on the blue rectangle (alpha<1) and the background is green.
5. Press “4”. Verify that only the green background is displayed.
6. Press “5”. Verify that the red rectangle (alpha<1) is overlayed on the blue rectangle (alpha=1) and the background is green.
7. Press “6”. Verify that both rectangles are black (alpha<1) and the background is green.
8. Press “7”. Verify that the intersection of the rectangles is red (alpha=1), the areas within the rectangles that do not intersect are black (alpha=1) and the background is green.
9. Press “8”. Verify that both rectangles are black (alpha<1) and the background is green.

## PropertiesTest

Three tests are available to verify the values of application properties. Each of the three xlets verifies a different security setting. The properties and their values are displayed on the screen with result summary as the first line.

Two of the three Permissions tests are run through the autoXlet method. Only the Unsigned ApplicationPropertiesTest is run manually. A description of all three properties tests is included below as well as instructions for how to run the UnsignedApplicationProperties Test manually.

**Unsigned Application Properties Test**

This is an unsigned application with AppId 3231.

**Signed Application Properties Test**

This is a signed application with AppId 4231.

**MonitorApplication Properties Test**

This is a dualy signed application with AppId 6231 and an associate PRF file granting it MonitorAppPermission(“properties”) so that the following properties can be verified:

* ocap.hardware.vendor\_id
* ocap.hardware.version\_id
* ocap.hardware.version
* ocap.hardware.model\_id
* ocap.hardware.serialnum
* ocap.hardware.createdate
* ocap.software.vendor\_id
* ocap.software.version
* ocap.software.model\_id
* ocap.memory.video
* ocap.memory.total

### Test Configuration

In $OCAPROOT/bin/$OCAPTC/env:

* Set final.properties to include:

*OCAP.xait.ignore=true*

* Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/ PropertiesTest / hostapp.properties.prop* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*

### Test Setup:

1. Edit hostapp.properties so the application\_control\_code for the application to be run is set to AUTOSTART (instead of PRESENT). In this case the UnsignedProperties App will be set to AUTOSTART.
2. From $PLATFORMROOT start the RI by entering ./*runRI.sh****.***
3. Any application values that are incorrect are marked with the word FAIL and the number of failure is displayed at top of screen. If test passes, the first line of the screen will indicate such.

## Time Test

This test application displays the time of the day clock. The time is updated every second.

### Test Configuration

In $OCAPROOT/bin/$OCAPTC/env:

* Set final.properties to include:

*OCAP.xait.ignore=true*

* Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/ TimeTest / hostapp.properties.time* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*

### Test Setup

1. From $PLATFORMROOT start the RI by entering ./*runRI.sh****.***
2. The time of day clock is displayed and is updated every second.

## Tune Test

This test application is a general tuning program. User can use the channel\_up and channel\_down keys to change channels. Depending on the config.properties settings, the test xlet can either build the channelMap dynamically calling SIManager.filterService(ServiceFilter) for non abstract services; or it can read in a static channelMap defined in the config.properties. By default, the channelMap is built based on the config.properties settings.

The TuneTest xlet can also be run in various automated modes by pressing keys 2, 3, or 5 to stress test the stack:

* Key 2: Tunes up and down with random waits between min\_delay and 15 seconds.
* Key 3: Tunes up and down with min\_delay between.
* Key 5: Tunes randomly through list at an increasing delay of 'interval' ms between min and max delay. Delay decreases once the max is reached and continues down the min, where it starts increasing again.

The values for min\_delay, max\_delay, and interval are set in config.properties. By default, they are set to:

min\_delay=5000

max\_delay=30000

interval=100

### Test Configuration

* Set final.properties to include:

OCAP.xait.ignore=true

* Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/TuneTest/hostapp.properties.tune* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*
* Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/TuneTest/config.properties.tune* to *$OCAPROOT/bin/$OCAPTC/env/qa/xlet/config.properties*

### Test Procedure

1. From $PLATFORMROOT start the RI by entering ./*runRI.sh****.***
2. User can now tune by in manual mode by pressing on the Channel\_UP/Down buttons, or go into auto mode by pressing on keys 2, 3, or 5.
3. User can also press key 1 to request the channelMap be re-read allowing the list specified in config.properties to be changed without restarting RI.

# Automated Xlet Tests

Below is the set of test xlets which are executed as part of the autoXlet test suite. The autoXlet test configuration and procedures and test descriptions are provided in the following sections. Also provided are instructions for manual execution of these tests. Note: These tests have only been run in the automated mode in the recent past. Manual setup instructions have been included for informational purposes but as of this writing these steps have not been re-verified.

* FileAccessPermissionTest - Create
* FileAccessPermissionTest - Read same org
* FileAccessPermissionTest - Write same org
* FileAccessPermissionTest - Read diff. org
* FileAccessPermissionTest - Write diff. org
* FileAccessPermissionTest - Remove diff. org
* FileAccessPermissionTest - Remove same org
* FocusTest
* FrontPanelResourceTest
* FrontPanel
* HSMScene
* PermissionTest - Signed withOUT PRF
* PermissionTest - Signed with PRF
* PermissionTest - Dually Signed withOUT PRF
* PermissionTest - Dually Signed with PRF
* PermissionTest - SuperHost (monitor app)
* PowerKeyTest
* PropertiesTest - Signed
* PropertiesTest - SuperHost (Monitor app)
* UserPreference
* DvrTest - Grp 1, tests 1-85

## Running the autoXlet test suite

### autoXlet Test Procedure

1. Make sure the following are defined in your $OCAPROOT/bin/$OCAPTC/env/

final.properties:

OCAP.xait.ignore=true

OCAP.fp.indicators=power record text message //for the FrontPanel test

1. Copy $OCAPROOT/apps/qa/host.properties.test to $OCAPROOT/bin/$OCAPTC/ env/hostapp.properties.
2. Copy $OCAPROOT/apps/qa/AutoXlet/config.properties.autoxlet to $OCAPROOT/bin/ $OCAPTC/env/qa/xlet/config.properties
3. Copy $OCAPROOT/apps/qa/AutoXlet/CL\_RI\_XletDriver.xml to $OCAPROOT/bin/ $OCAPTC/env/qa/xlet/xletDriver.xml.
4. Edit your $PLATFORMROOT/platform.cfg to turn on additional front panel LEDs for the FrontPanel test:

RI.Platform.frontpanel.number\_of\_leds = 3 // default is 2

RI.Platform.frontpanel.led.1.name = record //default is remote

# Add the following

RI.Platform.frontpanel.led.2.blue = FALSE

RI.Platform.frontpanel.led.2.green = FALSE

RI.Platform.frontpanel.led.2.yellow = FALSE

RI.Platform.frontpanel.led.2.orange = FALSE

RI.Platform.frontpanel.led.2.red = TRUE

RI.Platform.frontpanel.led.2.brightnesses = 2

RI.Platform.frontpanel.led.2.maxBlinksPerMinute = 240

RI.Platform.frontpanel.led.2.name = message

RI.Platform.frontpanel.led.3.blue = FALSE

RI.Platform.frontpanel.led.3.green = FALSE

RI.Platform.frontpanel.led.3.yellow = FALSE

RI.Platform.frontpanel.led.3.orange = FALSE

RI.Platform.frontpanel.led.3.red = TRUE

RI.Platform.frontpanel.led.3.brightnesses = 2

RI.Platform.frontpanel.led.3.maxBlinksPerMinute = 240

RI.Platform.frontpanel.led.3.name = remote

Also, for the DvrTestRunner tests, you need to have the platform.cfg file set up with 2 tuners:

RI.Platform.numTuners = 2

1. To run the Integration tests, simply start RI as you normally would using the ./runRI.sh script. Since xletDriver is the AUTOSTART xlet specified in hostapp.properties, it will then read in your config.properties which tells it to run through all the test xlets specified in CL\_RI\_XletDriver.xml.
2. Once the tests are done, you can find the results recorded in $OCAPROOT/bin/$OCAPTC/env/persistent/usr/1/7000/Results.txt. For DVRTestRunner tests, results are more easily checked by searching for the string "result=" in the AutoXlet debug log which is located at $OCAPROOT/bin/$OCAPTC/env/persistent/usr/1/7000/Debug.txt.

## File Access Permission Tests

Seven applications are available to test the permissions associated with files that are held in persistent storage.

The seven applications are configured through the hostapp.properties file to report whether operations pass or fail as expected. The results are presented to the screen. Results presented in GREEN indicate that the test has passed as expected. Results presented in ORANGE indicate that the test has failed.

The "FAP Create Files" application creates a set of files and then sets different FileAccessPermissions on them. It sets one with only owner rights (app1.txt), one with same-org rights (org1.txt), one with world rights (world1.txt), and one with other-org rights (other1.txt). If the permissions given to the files don't come back as expected, then the operation will produce a failure. This application also set the FileAccessPermission on the directory under which the four files are created to have same-org read and write rights.

There are two applications within the same organization that test whether or not they can access those files. These two applications are the "FAP Read Files (same org)" and "FAP Write Files (same org)".

The application “FAP Read Files (other org)”, “FAP Write Files (other org)”, and “FAP Remove Files (other org)” which has OrgId 2 (instead of 1) test whether access to the four files and their parent directory has been greated.

“FAP remove files (same org)” should be run last to verify the four created files can be deleted.

**FAP Create Application Test:**

This application should be run first as it creates and sets up the files needed for subsequent FileAccessPermission test. If the xlet run successfully, the following 4 files will be created under ***$OCAPROOT/bin/$OCAPTC/env/persistent/usr/1/7011/***:

1. app1.txt owner read and write permission
2. org1.txt same org (OrgId=1) read and write permission
3. world1.txt world read and write permission
4. other1.txt different org (OrgId=2) read and write permission

**FAP Read Same Org application Test**

This application should be run after FAP Create has run successfully. This xlet has the same OrgId as FAP Create. It verifies that it has read access to org1.txt and world1.txt while attempts to read app1.txt and other1.txt are denied.

**FAP Read Other Org application Test**

This application should be run after FAP Create has run successfully. This xlet has a different OrgId from FAP Create. It verifies that it has read access to world1.txt and other1.txt while attempts to read app1.txt and org1.txt are denied.

**FAP Write Same Org application Test**

This application should be run after FAP Create has run successfully. This xlet has the same OrgId as FAP Create. It verifies that it has write access to org1.txt and world1.txt while attempts to write to app1.txt and other1.txt are denied.

**FAP Write Other Org application Test**

This application should be run after FAP Create has run successfully. This xlet has a different OrgId as FAP Create It verifies that it has write access to world1.txt and other1.txt while attempts to read app1.txt and org1.txt are denied.

**FAP Remove Other Org Application Test**

This application should be run after FAP Create has run successfully. This xlet is a dually signed application (AppId 7016) with a different OrgId as FAP Create. It verifies that it does not have write access to the parent directory under which the four files were created, hence it can not delete the four files created by “FAP create”.

**FAP Remove Application Test**

This application should be run after FAP Create has run successfully. This xlet is a dually signed application (AppId 7015) with the same OrgId as FAP Create. It verifies that it has write access to the parent directory under which the four files were created, hence it can delete the four files created by “FAP”.

### Test Setup

These tests are run as part of the autoXlet test suite. See section 3.2 for instruction on how to run the autoXlet test suite. To run these tests manually follow the instructions below:

1. Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/FileAccessPermissionTest/hostapp.properties.fap* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*
2. Edit hostapp.properties so the application\_control\_code for app.0 (FAP Create Fiels) is set to AUTOSTART (instead of PRESENT).
3. Start RI by running ***runRI.sh –deletestorage***
4. If the application was successful, the output is in green and necessary files will be created. the following 4 files will be created.
5. Exit RI (Ctrl-C)
6. Edit hostapp.properties so the application\_control\_code for app.1 (FAP Read Same Org) is the only one set to AUTOSTART (instead of PRESENT). (application\_control\_code for all other apps should be set to PRESENT).
7. Restart RI by running ***runRI.sh***
8. If the test passes, the output is in green.
9. Repeat steps 5-8 for app.2 (FAP Read other org), app.4 (FAP write same org), and app.3 (FAP remove).

## Focus Test

This is a basic xlet to test Havi HScene and HComponent objects. The xlet manages 4 other xlets that have coordinated sections of the screen. Each xlet has the ability to grab focus of the screen and controls, giving the user the ablity to select components with the graphic, give focus to the scene or components, and set visibility or active status of the scene. Xlets can be start, paused, or destroyed as many times as needed.

To run FocusTestRunnerXlet:

Up and down arrows - select between each of the 4 displayed xlets

Left and Right arrows - select between selection of the scene or text box components

1 - requestFocus() - give the specified component or scene focus. Text box components will be highlighted once they are given focus

2 - setVisible(true) - Set the scene attribute such that the grphics for the xlet are shown

3 - setVisible(false) - Set the scene attribute such that the grphics for the xlet are hidden

4 - setActive(true) - Set whether the HScene is prepared to accept focus

5 - setActive(false) - Set to cause the scene to lose or not accept focus

Play - Starts the Xlet

Stop - Stops the Xlet

Pause - Pauses the Xlet

### Test Procedure

These tests are run as part of the autoXlet test suite. See section 3.2 for instruction on how to run the autoXlet test suite. To run these tests manually follow the instructions below:

1. Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/FocusTest/ hostapp.Focustest* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties.*
2. Start RI by running ***runRI.sh***.
3. Start Focus 1 - Press Play.
4. Select Focus1-Text2.
5. Press 1 to request focus on the text.
6. Select Focus1-Text1.
7. Press 1 to request focus on the text.
8. Select Focus 4.
9. Start Focus 4 - Press Play.
10. Press 1 to request focus on the text.
11. Select Focus 4 by pressing left arrow.
12. Press 3 to set the visibility of the scene to false.
13. Select Focus 3.
14. Start Focus 3 - Press Play.
15. Press 1 to request focus on the scene.
16. Select Focus 4.
17. Press 2 to set the visibility of the scene to true.
18. Press 1 to request focus on the scene.
19. Select Focus 3.
20. Press 2 to set the visibility of the scene to true.
21. Press 1 to request focus on the text.

## FrontPanelExt Test

The FrontPanelExtTest xlet tests the following settop box front panel indicators: POWER, RECORD, MESSAGE, RFBYPASS, TEXT DISPLAY, and CLOCK.

For the TEXT DISPLAY tests, "text" should be included in the space-separated list of values for the OCAP.fp.indicators property in the mpeenv.ini file (for example: OCAP.fp.indicators=message power text).

SA8300HD Note: Although the PowerTV API supports a range of brightness levels from 0x0001 through 0xFFFF, the SA8300HD appears to support only ON (0xFFFF) and OFF (any value < 0xFFFF).

### TEST PROCEDURE

These tests are run as part of the autoXlet test suite. See section 3.2 for instruction on how to run the autoXlet test suite. To run these tests manually follow the instructions below:

1. Create a hostapp.properties file for the FrontPanelExt app and copy it to $OCAPROOT/bin/$OCAPTC/env/hostapp.properties.

Sample hostapp.properties parameters:

app.57.application\_identifier=0x000000016619

app.57.application\_control\_code=PRESENT

app.57.visibility=VISIBLE

app.57.priority=220

app.57.application\_name=FrontPanelExt

app.57.base\_directory=/snfs/qa/xlet

app.57.initial\_class\_name=org.cablelabs.xlet.FrontPanelExtTest.FrontPanelExtXlet

app.57.service=0x12355

1. Start RI by running runRI.sh.
2. The following list of test cases is available. To run a test case, key in the test case number, then press ENTER.

Press A (yellow trianble) for current status of all available indicators

Press B (blue square) for current status of Text Display

Test 0 : toggle POWER indicator reservation

Test 1 : toggle RECORD indicator reservation

Test 2 : toggle MESSAGE indicator reservation

Test 3 : toggle RFBYPASS indicator reservation

Test 4 : toggle TEXT DISPLAY reservation

Test 5 : reset POWER indicator blink spec (rate)

Test 6 : reset POWER indicator blink spec (duration)

Test 7 : reset POWER indicator bright spec

Test 8 : reset POWER indicator color spec

Test 9 : reset RECORD indicator blink spec (rate)

Test 10 : reset RECORD indicator blink spec (duration)

Test 11 : reset RECORD indicator bright spec

Test 12 : reset RECORD indicator color spec

Test 13 : reset MESSAGE indicator blink spec (rate)

Test 14 : reset MESSAGE indicator blink spec (duration)

Test 15 : reset MESSAGE indicator bright spec

Test 16 : reset MESSAGE indicator color spec

Test 17 : reset RFBYPASS indicator blink spec (rate)

Test 18 : reset RFBYPASS indicator blink spec (duration)

Test 19 : reset RFBYPASS indicator bright spec

Test 20 : reset RFBYPASS indicator color spec

Test 21 : reset TEXT DISPLAY text string setting

Test 22 : erase text from text display

Test 23 : toggle TEXT DISPLAY text wrap setting

Test 24 : reset TEXT DISPLAY blink spec (rate)

Test 25 : reset TEXT DISPLAY blink spec (duration)

Test 26 : reset TEXT DISPLAY bright spec

Test 27 : reset TEXT DISPLAY color spe

Test 28 : reset TEXT DISPLAY scroll spec (horizontal rate)

Test 29 : reset TEXT DISPLAY scroll spec (vertical rate)

Test 30 : reset TEXT DISPLAY scroll spec (hold duration)

Test 31 : reset TEXT DISPLAY CLOCK mode setting

Test 32 : reset TEXT DISPLAY CLOCK blink spec (rate)

Test 33 : reset TEXT DISPLAY CLOCK blink spec (duration)

Test 34 : reset TEXT DISPLAY CLOCK bright spec

Test 35 : reset TEXT DISPLAY CLOCK color spec

Test 36 : reset TEXT DISPLAY CLOCK scroll spec (horizontal rate)

Test 37 : reset TEXT DISPLAY CLOCK scroll spec (vertical rate)

Test 38 : reset TEXT DISPLAY CLOCK scroll spec (hold duration)

1. Evaluate test results by viewing the TV screen and the console log.

## FrontPanelResource Test

The Front Panel Resource test application exercises allocation of resources among multiple applications and resolving resource contention. The test application can launch up to four additional applications with varying priorities. Each of these applications can then be set to reserve or release resources and indicate whether or not they are willing to give up a resource when there is contention. There are two front panel components that an application can reserve, text or message component. Depending on an applications priority, it can lose a resource even if it is unwilling because an application with higher priority will be given the resource when it requests it. The Front Panel Resource test application verifies that resource contention is resolved properly.

The user interface of the application allows for:

* Starting, stopping, or pausing any of the four different applications
* Reserving or releasing a front panel component for an application
* Making an application willing to release a component
* Printing out the test results

**User Interface**

When the application is loaded, a test launcher will be presented on the screen. Selecting FPResourceTestRunner, will launch the test application.

Control Keys

UP ARROW - Scroll up through the four xlets

DOWN ARROW - Scroll down through the four xlets

LEFT ARROW - Change through the different Front Panel components

RIGHT ARROW - Change through the different Front Panel components

PLAY - Start currently selected xlet

STOP - Stop currently selected xlet

PAUSE - Pause currently selected xlet

1 - Reserve indicator for current xlet

2 - Release indicator for current xlet

3 - Toggle willing to release resource for current xlet

0 - Print test results to console

### TEST PROCEDURE

These tests are run as part of the autoXlet test suite. See section 3.2 for instruction on how to run the autoXlet test suite. To run these tests manually follow the instructions below:

1. Application properties for the test should be defined in the hostapp.properties file. The application contains a hostapp.properties file that should be used for launching this application.

For example:

###############################################

## Application 1 - Front Panel Resource Test

###############################################

app.1.application\_identifier=0x000000016610

app.1.application\_control\_code=PRESENT

app.1.visibility=VISIBLE

app.1.priority=220

app.1.application\_name=FPResourceTestRunner

app.1.base\_directory=/snfs/qa/xlet

app.1.initial\_class\_name=org.cablelabs.xlet.FrontPanelResourceTest.FPTestRunnerXlet

app.1.service=0x12355

app.1.args.0=0x000000016611

app.1.args.1=0x000000016612

app.1.args.2=0x000000016613

app.1.args.3=0x000000016614

###############################################

###############################################

## Application 2 - Front Panel Resource Test

###############################################

app.2.application\_identifier=0x000000016611

app.2.application\_control\_code=PRESENT

app.2.visibility=INVISIBLE

app.2.priority=220

app.2.application\_name=FPTest1

app.2.base\_directory=/snfs/qa/xlet

app.2.initial\_class\_name=org.cablelabs.xlet.FrontPanelResourceTest.FPTestXlet

app.2.service=0x12355

app.2.args.0=x=0

app.2.args.1=y=0

app.2.args.2=width=213

app.2.args.3=height=240

app.2.args.4=runner=0x000000016610

###############################################

###############################################

## Application 3 - Front Panel Resource Test

###############################################

app.3.application\_identifier=0x000000016612

app.3.application\_control\_code=PRESENT

app.3.visibility=INVISIBLE

app.3.priority=220

app.3.application\_name=FPTest2

app.3.base\_directory=/snfs/qa/xlet

app.3.initial\_class\_name=org.cablelabs.xlet.FrontPanelResourceTest.FPTestXlet

app.3.service=0x12355

app.3.args.0=x=213

app.3.args.1=y=0

app.3.args.2=width=213

app.3.args.3=height=240

app.3.args.4=runner=0x000000016610

###############################################

###############################################

## Application 4 - Front Panel Resource Test

###############################################

app.4.application\_identifier=0x000000016613

app.4.application\_control\_code=PRESENT

app.4.visibility=INVISIBLE

app.4.priority=220

app.4.application\_name=FPTest3

app.4.base\_directory=/snfs/qa/xlet

app.4.initial\_class\_name=org.cablelabs.xlet.FrontPanelResourceTest.FPTestXlet

app.4.service=0x12355

app.4.args.0=x=0

app.4.args.1=y=240

app.4.args.2=width=213

app.4.args.3=height=240

app.4.args.4=runner=0x000000016610

###############################################

###############################################

## Application 5 - Front Panel Resource Test

###############################################

app.5.application\_identifier=0x000000016614

app.5.application\_control\_code=PRESENT

app.5.visibility=INVISIBLE

app.5.priority=220

app.5.application\_name=FPTest4

app.5.base\_directory=/snfs/qa/xlet

app.5.initial\_class\_name=org.cablelabs.xlet.FrontPanelResourceTest.FPTestXlet

app.5.service=0x12355

app.5.args.0=x=213

app.5.args.1=y=240

app.5.args.2=width=213

app.5.args.3=height=240

app.5.args.4=runner=0x000000016610

###############################################

NOTE: This application has been designed so that it can be set up to run in automation. The hostapp.properties file that is supplied with the application must be modified so that the XletDriver is enabled.

1. The application can be evaluated by viewing the screen and the console logging to verify that the test results are correct after running the application and dumping the test results to the console.

## HSMScene Test

The HSceneManagerTest xlet tests the org.ocap.ui.HSceneManager API. The test provides menu options to perform the following tasks:

* Create and run multiple test xlets.
* Change the z-ordering of the xlets' scenes on the TV screen.
* Change the position and the size of the xlets' scenes.

### Test Procedure

These tests are run as part of the autoXlet test suite. See section 3.2 for instruction on how to run the autoXlet test suite. To run these tests manually follow the instructions below:

1. The hostapp.properties file should define the test xlets to be created by the HSceneManagerTest.

SAMPLE HOSTAPP.PROPERTIES PARAMETERS:

###############################################

## HSceneManagerTest

###############################################

app.1.application\_identifier=0x000000016670

app.1.application\_control\_code=PRESENT

app.1.visibility=VISIBLE

app.1.priority=220

app.1.application\_name=HSMTestRunner

app.1.base\_directory=/snfs/qa/xlet

app.1.initial\_class\_name=org.cablelabs.xlet.HSceneManagerTest.HSMTestRunnerXlet

app.1.args.0=testXlet1=0x000000014671

app.1.args.1=testXlet2=0x000000014672

app.1.args.2=testXlet3=0x000000014673

app.1.args.3=testXlet4=0x000000014674

app.1.args.4=testXlet5=0x000000014675

app.1.args.5=testXlet6=0x000000014676

app.1.args.6=noShow=0x000000014675

app.1.args.7=noOrder=0x000000014674

app.1.args.8=noMove=0x000000014673

###############################################

app.2.application\_identifier=0x000000014671

app.2.application\_control\_code=PRESENT

app.2.visibility=VISIBLE

app.2.priority=100

app.2.application\_name=HSMTest1

app.2.base\_directory=/snfs/qa/xlet

app.2.initial\_class\_name=org.cablelabs.xlet.HSceneManagerTest.HSMTestXlet

app.2.args.0=x=25

app.2.args.1=y=25

app.2.args.2=width=250

app.2.args.3=height=250

app.2.args.4=runner=0x000000016670

app.2.args.5=color=green

###############################################

app.3.application\_identifier=0x000000014672

app.3.application\_control\_code=PRESENT

app.3.visibility=VISIBLE

app.3.priority=150

app.3.application\_name=HSMTest2

app.3.base\_directory=/snfs/qa/xlet

app.3.initial\_class\_name=org.cablelabs.xlet.HSceneManagerTest.HSMTestXlet

app.3.args.0=x=50

app.3.args.1=y=50

app.3.args.2=width=250

app.3.args.3=height=250

app.3.args.4=runner=0x000000016670

app.3.args.5=color=cyan

###############################################

app.4.application\_identifier=0x000000014673

app.4.application\_control\_code=PRESENT

app.4.visibility=VISIBLE

app.4.priority=150

app.4.application\_name=HSMTest3

app.4.base\_directory=/snfs/qa/xlet

app.4.initial\_class\_name=org.cablelabs.xlet.HSceneManagerTest.HSMTestXlet

app.4.args.0=x=75

app.4.args.1=y=75

app.4.args.2=width=250

app.4.args.3=height=250

app.4.args.4=runner=0x000000016670

app.4.args.5=color=orange

###############################################

1. The HSceneManagerTest provides the following menu options:

<< CH UP / CH DOWN >> Change Current Xlet

(PLAY) Start Xlet

(STOP) Stop Xlet

(PAUSE) Pause Xlet

(1) Toggle Move Type (between Size and Position)

(2) Pop to Front (brings the currently selected xlet's scene to the front)

(UP) Decrease Y Size/Position

(DN) Increase Y Size/Position

(LT) Decrease X Size/Position

(RT) Increase X Size/Position

1. Evaluate by viewing the TV screen and the console log. The z-order of the test xlets is printed to the TV screen and to the console log.

## PermissionTest

Six applications are available to check that the correct permissions are granted or denied based on the application's AppId and associated Permission Request File (PRF). One of the applications is run manually the remainder are run as part of the autoXlet test suite. Please see section 2.4 for a complete description of all PermissionTest applications.

## PowerKey Test

This test application exercises the power key event listener by allowing user to register and unregister event listeners by calling host.addPowerModeChangeListener(…) and host.removePowerModeChangeListener(…) respectively.

### Test Procedure

These tests are run as part of the autoXlet test suite. See section 3.2 for instruction on how to run the autoXlet test suite. To run these tests manually follow the instructions below:

1. Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/PowerKeyTest/ hostapp.properties.power* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*
2. Start RI by running ***runRI.sh***.
3. Once the application has started, press the “Power” key a few time and see the power mode toggle. Verify asynchronous power change event should not have been received.
4. Press “1” to register the PowerModeChange listener.
5. Press “Power” key a few times again and see the value toggle. Also verify the asynchronous event are received.
6. Press “2” to unregister the listener.
7. Press “Power” key a few times and see the value continue to toggle between Full and Low. Verify that event are again not received.
8. Press “3” at any time and see the current/actual state is retrieved and displayed via the synchronous call.

## PropertiesTest

Three tests are available to verify the values of application properties. One of the applications is run manually the remainder are run as part of the autoXlet test suite. Please see section 2.6 for a complete description of all PropertiesTest applications.

## UserPreference Test

This application tests the ability to read, clear, set and write user preferences to the UserPrefMgr.

### Test Procedure

1. Copy $OCAPROOT/apps/qa/org/cablelabs/xlet/UserPrefs/ hostapp.properties.userPref to $OCAPROOT/bin/$OCAPTC/env/hostapp.properties.
2. Start RI by running runRI.sh

## DVR Test Runner Test

The DVRTestRunner xlet contains a collection of hundreds of tests which exercise the DVR stack. All of the tests contained within the DVR test runner xlet utilize an array of locators for their various tuning parameters. The locators are specified in the config.properties. It is required to have 5 channels listed in DVRTestRunner to have it work properly. It is optional to have the frequency, program#, QAM. If DVR\_by\_FPQ is set to true, DVR\_FPQ\_# values will be used. Otherwise DVR\_sourceId\_# will be used.

### Test Procedure

These tests are run as part of the autoXlet test suite. See section 3.2 for instruction on how to run the autoXlet test suite. To run these tests manually follow the instructions below:

1. Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/DvrTest/ hostapp.properties.dvrTestRunner* to *$OCAPROOT/bin/$OCAPTC/env/hostapp.properties*
2. Copy *$OCAPROOT/apps/qa/org/cablelabs/xlet/DvrTest/ config.properties.dvrTestRunner* to *$OCAPROOT/bin/$OCAPTC/env/qa/xlet/config.properties*
3. Start RI by running ***runRI.sh***
4. User can now choose to run the test they are interested in by entering the group number followed by <ENTER> then the test number followed by <ENTER>.

With this release (1.1.4-C), tests 1-85 in group1 have been cleaned up and validated:

Test 3-4 TestBasicRecordingAndPlayback.TestScheduledRecording():

* Schedules a 30 seconds recording to start 30 seconds from now.
* At the end of recording (60secs), print out all recordings.
* test3 records a digital service; test 4 records an analog service

Test verifications:

* Verifies recording is in the COMPLETED\_STATE (70 seconds)
* Verifies that the recording changed state 3 times - recordingChanged()was called 3 times:
  + enter PENDING\_NO\_CONFLICT\_STATE when the entry was first added
  + PENDING\_NO\_CONFLICT\_STATE to IN\_PROGRESS\_STATE
  + IN\_PROGRESS\_STATE to COMPLETE\_STATE

Test 5 TestBasicRecordingAndPlayback.TestRecordingPlayback():

* initialize ServiceContext
* Schedule a 30 seconds recording to start in 2 seconds
* after recording is done (41 seconds), select the recorded service
* wait 30 seconds before cleaning up the ServiceContext
* (tune to the RecordingRequest's service using ServiceContext.select()

Test verification:

* Verifies recording is in the COMPLETED\_STATE (40 seconds)
* Verifies no issues with presenting the recorded service

(NormalContentEvent was received).

Test 6 TestBasicRecordingAndPlayback.TestRecordingPlaybackDeleteInProgress():

* initialize ServiceContext
* Schedule a 40 seconds recording to start in 2 seconds
* select the recorded service while recording is in progress (12 seconds)
* Instead of waiting blindly for 30 seconds, wait for the

PresentationTerminatedEvent

(tune to the RecordingRequest's service using ServiceContext.select()

* Delete the RecordingRequest while it is in progress (30 seconds)

Test verification:

* Verifies recording gets to in the IN\_PROGRESS\_STATE (19 seconds)
* Verifies no issues with presenting the recorded service (NormalContentEvent was received then the PresentationTerminatedEvent and ServiceContext is cleaned up successfully).

Test 7 TestBasicRecordingAndPlayback.TestRecordingPlaybackUsingJMF():

Test 8 TestBasicRecordingAndPlayback.TestRecordingPlaybackSetMediaTime():

Test 9 TestConsecutiveRecordings.TestBasicConsecutiveRecordings()

* Schedule eight 1-minute recordings back to back.

- "Recording1" to start 1min from now and ends 2min from now

- "Recording2" to start 2min from now and ends 3min from now

- "Recording3" to start 3min from now and ends 4min from now

- "Recording4" to start 4min from now and ends 5min from now

- "Recording5" to start 5min from now and ends 6min from now

- "Recording6" to start 6min from now and ends 7min from now

- "Recording7" to start 7min from now and ends 8min from now

- "Recording8" to start 8min from now and ends 9min from now

Test verification:

* verify all eight recordings reached the COMPLETED\_STATE 10mins from now

Test 10 TestConsecutiveRecordings.TestBasicConsecutiveRecordings2()

* Schedule two 40 second recordings 10 seconds apart on different locators
  + "Recording1" to start 50 seconds from now and ends 1.5 minutes from now
  + "Recording2" to start 1min40seconds from now and ends 2min20sec from now

Test verification:

* verify both recordings reached the COMPLETED\_STATE 2min30secs from now

Test 11 TestConsecutiveRecordings.TestSimultaneousRecordings()

* Schedule four pairs of 45 secs recordings on the same locator 1 min apart
  + Recording1 & Recording2 starts 10secs from now and ends 55secs from now
  + Recording3 & Recording4 starts at 1min10secs and ends at 1min55secs
  + Recording5 & Recording6 starts at 2min10secs and ends at 2min55secs
  + Recording7 & Recording8 starts at 3min10secs and ends at 3min55secs

Test verification:

* verify all eight recordings reached the COMPLETED\_STATE 4mins from now

Test 12, 13 TestImmediateRecording.TestBasicTSRecording()

* Select a service via ServiceContext
* at 1min10seconds from now, schedule a 30 secs long recording (by ServiceContext), "Recording1", in the past (starts at 30secs and ends at 1min).
* at 1min23secs from now, stop the broadcast
* at 1min32secs from now, select "Recording1".

Test verification:

* Verify recording reached the COMPLETED\_STATE at 1min15secs (before broadcast was stopped).
* Verify "Recording1" is still in the COMPLETED\_STATE at 2min10secs (after playback of "Recording1" is complete".

Test 14, 15 TestImmediateRecording.TestImmediateTSRecording()

* Select a service via ServiceContext
* at 50seconds from now, schedule a 30 secs long recording (by ServiceContext), "Recording1", to start NOW which mean "Recording1" will go from 50secs to 1min20secs.
* at 1min23secs from now (after the recording is done), stop the broadcast
* at 1min32secs from now, select "Recording1".

Test verification:

* Verify recording reached the COMPLETED\_STATE at 1min22secs (before broadcast was stopped).
* Verify "Recording1" is still in the COMPLETED\_STATE at 2min10secs (after playback of "Recording1" is complete".

Test 16 TestImmediateRecording.TestSCRecordingTuneAway()

* Select a service (loc0) via ServiceContext
* at 40seconds from now, schedule a 6min long recording (by ServiceContext), "Recording1", of loc0 to start 30 seconds from now which means "Recording1" will complete at 6min30secs.
* at 1min10secs from now (while "Recording1" is IN\_PROGRESS), select a different service (loc1) via ServiceContext.
* at 2min5secs from now, stop the broadcast
* at 2min20secs from now, select "Recording1"

Test verification:

* Verify "Recording1" is in the INCOMPLETE\_STATE after the service has been tuned away (at 1min25secs and 2min30secs)

Test 17 TestImmediateRecording.TestTSRecordingSchedRecording()

* Select a service via ServiceContext
* at 30seconds from now, schedule a 2min20secs long recording (by ServiceContext), "SCRecording1", to start NOW which mean "SCRecording1" will go from 30secs to 2min50secs.
* at 1min from now, schedule a 30 secs long recording, "Recording1", to start NOW which means "Recording1" will go from 1min to 1min30secs. "Recording1" overlaps the middle section of "SCRecording1"

Test verification:

* at 1min50secs, verify "Recording1" is in COMPLETE\_STATE and "SCRecording1" is in "IN\_PROGRESS\_STATE"
* after 3mins (post end time of "SCRecording1"), verify both recordings are in COMPLETE\_STATE and both recordings can be played back successfully

Test 18 TestOverlappingEntries.TestOverlappingEntries()

* Schedule 12 recordings to start the next day and verify RecordingRequest.getOverLappingEntries() returns correctly.
* None of the recording actually execute, all recordings are in the PENDING state

Test verification

* Verify getOverLappingEntries() for "Recording9" returned correctly:
  + there should be 8 overlapping recordings (0, 1, 2, 3, 4, 5, 7, 8)

Test 19, 20, 21 TestOverlappingRecordings.TestOverlappingSchedRecs()

* Schedule two 90 seconds recordings to overlap by 40 seconds:
  + "Recording1" starts 10 seconds from now and ends 1min40secs from now
  + "Recording2" starts 1 minute from now and ends 2min30secs from now

Test verification

* Verify both recordings reached the COMPLETED\_STATE 2 min40secs from now

Test 22, 23 TestOverlappingRecordings.TestOverlappingRecsWithSC()

* Start up a buffering service context
* Schedule two 60 seconds recordings to overlap by 30 seconds:
  + "Recording1" starts 1minute from now and ends 2minutes from now
  + "Recording2" starts 30 seconds from now and ends 1min30sec from now

Test verification

* Verify both recordings reached the COMPLETED\_STATE 2 min20secs from now

Test 24 TestRecordAndCancel.TestBasicRecordAndCancel()

* Schedule an one hour long recording "Recording1" to start in one hour
* Cancel Recording1 before it starts.
* Clean up the test by deleting Recording1 after it has been canceled

Test verification

* Verify "Recording1" reached the correct states: PENDING\_NO\_CONFLICT, then CANCELLED\_STATE.

Test 25 TestRecordAndCancel.TestBasicSchedule()

* Schedule five one hour long recordings to start in one day.

Test verification

* Verify the number of entries reported by RecordingManager increases by 5 the five recordings have been scheduled.

Test 26, 27 TestRecordingAlertLisener.TestBasicRecordingAlertListener()

* Schedule 3 recordings
* Test 30 deletes one of the PENDING recordings seconds before it is

scheduled to start.

Test verification

* Verify the correct number of RecordingAlertEvent is received.

Test 28, 29 TestRecordingPlaybackListener.TestSingleListnerCallback()

* Register a RecordingPlabyckListener
* Create a recording "Recording1" and then playback once (test 31) or twice (Test 32).

Test verification

* \*Verify notifyRecordingPlayback(...) was called the expected number of times
  + once for test 31; twice for test 32.

Test 30, 31 TestRecordingPlaybackListener.TestMultiListnerCallback()

* Register 100 RecordingPlabyckListener to the OcapRecordingManager (each listener is registered to ORM twice
* Create a recording "Recording1" and then playback once (test 33) or twice (Test 34).

Test verification

* Verify notifyRecordingPlayback(...) was called the expected number of times for each of the 100 listeners.
  + once for test 33; twice for test 34.

Test 32 TestRecordingID\_ECN829.TestRecordingIDBasic()

* Verify that an IllegalArgumentException is thrown if getRecordingRequest(...) is called with a non-existing id.
* Verify all existing recordings ID (retrieve by calling getId() are unique
* schdule 8 recordings

Test verification

* Verify RecordingRequest.getId() returns a unique ID for each of the 8 recordings
* Verify RecordingManager.getRecordingRequest(...) returns valid RecordingRequest for all ids (existing recoridng as well as the 8 newly added.

Test 33-38 TestRecMgrGetEntries.TestRecMgrGetEntriesFilter()

* Schedules three 30 seconds recordings that overlap eachother:
  + Recording1a, Recording1b, Recording1c starts 10 secs after eachother
* Also schedules five 30 seconds back-to-back recordings (Recording2-6) with the first of these recordings (Recording2) starting 10 seconds after Recording1c's end time.

Test Verification

* Verifies RecordingManager.getEntries(RecordingStateFilter) is correctly implemented by confirming the expected RecordingList is returned for various States:
* Test 33: PENDING\_NO\_CONFLICT\_STATE
* Test 34: PENDING\_WITH\_CONFLICT\_STATE
* Test 35: IN\_PROGRESS\_STATE
* Test 36: INCOMPLETE\_STATE
* Test 37: FAILED\_STATE
* Test 38: COMPLETED\_STATE

Test 39 TestSortRecordinList.TestSortRecordingListTimeOrder()

* Schedule eight 90 seconds long recording in the far future. The start time of each recording are out of order.
* Implement a RecordingListComparator that sorts on the recording's start time

Test Verification

* Verify the RecordingList returned by the sortRecordingList() call is in order based on the Recording's start time

Test 40, 41 TestInsufficientStateChange.TestRecordWithInsufficientSpace()

* Schedule 2 large recordings:
  + duration of Rec1 takes up almost all the remaining disk space
  + duration of Rec2Helper can not fit on the remaining disk space

Test Verification

* Verifies that Rec1 transitions from PENDING\_NO\_CONFLICT\_STATE to IN\_PROGRESS\_STATE to IN\_PROGRESS\_INSUFFICIENT\_SPACE\_STATE when Rec2Helper starts.
  + test 40: Rec2Helper gets deleted which then causes Rec1 to transition back to the IN\_PROGRESS\_STATE
* Verifies that Rec2Helper transitions from PENDING\_NO\_CONFLICT\_STATE to IN\_PROGRESS\_INSUFFICIENT\_SPACE\_STATE
  + test 41: Rec2Helper does not gets deleted and will then transition to INCOMPLETE\_STATE due to SPACE\_FULL

Test 42, 43 TestScheduledRecordingContention.TestSimpleScheduledRecordingContention

* Schedules three 3-minutes long parallel RecordingRequests:
  + rch3-recFirst: scheduled first
  + rch2-recMiddle: scheduled second
  + rch1-recLast: scheduledLast
* Test 42: does not have a ResourceContentionHandler registered
* Test 43: has a default ResourceContentionHandler that prioritizes recordings by name lexigraphically registered

TestVerification

* Verify recordings are prioritized correctly:
* Test 42 (no RCH): rch3-recFirst and rch2\_recMiddle gets the resources and transitions from PENDING\_NO\_CONFLICT to IN\_PROGRESS while rch1-recLast transition from PENDING\_WITH\_CONFLICT to FAILED
* Test 43 (w/ RCH): rch1-recLast and rch2\_recMiddle gets the resources and transitions from PENDING\_NO\_CONFLICT to IN\_PROGRESS while rch3-recFirst transition from PENDING\_WITH\_CONFLICT to FAILED

Test 44-46 TestScheduledRecordingContention.TestReprioritizedContention\*

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedule a higher priority RecordingRequest (R1) after 4 recordings requests (R2-R5) have already been scheduled:

~ R4 and R5 are parallel

~ R2 end when R4/5 starts

~ R3 spans R2 and R4/5

Test 44: --------------------------------

-R4-

-R2-

-R5-

---R3---

-R1- scheduled while R2-R5 are in PENDING\*State

Test 45: --------------------------------

-R4-

-R2-

-R5-

---R3---

-R1- scheduled while R2-R5 are in PENDING\*State

Test 46: same as 48 except R1 is scheduled after R3 has started but R2 is still in the PENDING\_NO\_CONFLICT\_STATE

Test Verification

* Verify resource contentions are resolved correctly where recordings are given priority according to their name.
* Verify each recording transitions through the correct recording states

Test 47-54 TestScheduledRecordingContention1.TestRRContention\*

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Then schedule multiple overlapping recordings. Then verify contention are resolved as expected by rescheduling/deleteing/cancelling one or more of the recordings

Test Verifications:

* Verify each recording transition to the correct recording stats
* Test 47, 48: four recordings with first three in parallel, and last one starts immediately after

Time Line -------------------

|-R1-|

|-R2-|

|-R3-|

|-R4-|

Test 47: reschedules/deletes/cancels R1 so it is parallel to R4

Test 48: reschedules/deletes/cancels R3 so it is parallel to R4

* Test 49, 50: four recordings with first two in parallel (R1, R2), and last two in series (R3, R4) and spans the first 2 recording

Time Line -------------------

|----R1---|

|----R2---|

|-R3-|-R4-|

* + Test 49: reschedules/deletes/cancels R1 so it starts after R4 ends
  + Test 50: reschedules/deletes/cancels R3 so it starts after R4 ends
* Test 51: firve recordings where R1&R2 and R4&R5 in series while R3 spans in the middle:

Time Line -------------------

|-R1-|

|-R2-|

|-R4-|

|-R5-|

|-R3-|

Reschedule/delete/cancel R3 to resolve contention

* Test 52: six recordings: 3 pairs of parallel recordings in series

Time Line -------------------

|-R1-|

|-R4-|

|-R2-|

|-R5-|

|-R6-|

|-R3-|

* Reschedule R3 to have it start earlier which causes contention with R5
* Reschedule R3 back to resolve contention
* Reschedule R3 again to have it start earlier and end later to cause contention with both R5 and R6
* Reschedule R3 again so it has the original start time so contention with R5 is resolved, still has contention with R6
* Delete R3 to resolve contention
* Create R3 again with the earlier start time so there is contention with R5 again.
* Cancel R3 to resolve contention
* Test 53, 54: Six recordings: R1&R2 in parallel, R3&R4 in parallel with each other and in series with R1&R2. R5 starts first and intersects with R1&R2. R6 intersects R1&R2 and R3&R4:

Time Line -------------------

|-R1-|

|-R2-|

|-R3-|

|-R4-|

|-R5-|

|-R6-|

Resolve contention by:

~ Reschedule R1 and R4 to start after R3 is done

~ Delete R1 and R4

~ Cancel R1 and R4

Test 55-58 TestAppData:TestAddAppData

* Schedule a RecordingRequest and then make various call rr.addAppData()
  + (test 55 and 57 schedules a LeafRecordingRequest)
  + (test 56 and 58 schedules a ParentRecordingRequest)
  + (test 57 & 58 creates the RecordingRequest with one AppData entry associated.

Test Verification

* Verifies that IllegalArgumentException is correctly thrown if AppData exceeds limit
* Verifies that addAppData() can supports up to 64 keys
* Verifies that NoMoreDataEntriesException is thrown by addAppDAta() if there are already 64 app data entries.
* Verifies AppData (keys and AppData) are correct and can be removed successfully.

Test 59 TestSetParent:TestSetParentRecording

* Schedules 4 RecordingRequests
  + orr1 and orr2 are both long OcapRecordingRequests to start tomorrow
  + prr1 and prr2 are both ParentRecordingRequests that will be set as parent of orr1 and orr2

Test Verification: OcapRecordingRequest.setParent():

* Verify ocapRecordingRequest does not have a parent when it is first created (getParent() should return null)
* Verify prr1 can be set as parent of orr1 correctly with prr1 transitioning to COMPLETED\_RESOLVED\_STATE
* Verify orr1 can be reset with prr2 as its parent and prr2 is transition to PARTIALLY\_RESOLVED\_STATE; verify prr1's state is transitioned (to PARTIALLY\_RESOLVED\_STATE) correctly as result
* Verify ISE is thrown when setting ParentRecordingRequest(prr2) in CANCELLED\_STATE as the parent of an OcapRecordingRequets (orr2)
* Verify orr1 can be reset with prr1 as its parent and prr2 is correctly deleted from the recording database as it is in the CANCELLED\_STATE and contains no other RecordingRequests.

Test 60-63 TestSeriesRecordingExpiration:TestSeriesRec\_Season

* Set up a series recording with a Root ParentRecordingRequest(Show) which has a child branch that is also a ParentRecordingRequest (Season) which has two parallel children that are LeafRecordingRequests (Episodes)
* Delete one of the LeafRecordingRequests (Episode2) while both leafs are IN\_PROGRESS
* Wait till Recoridng expiration time has run out.
  + Test 60: 24 hours expiration time - long expiration time
  + Test 61: 90 sec expiration time - expires after recording's end time
  + Test 62: 1 minute expiration time - expires at recording's end time
  + Test 63: 30 sec expiration time - expired before recording's end time

Test Verification:

* Verify the DeletaionDetails of Episode2 shows it was "USER\_DELETED"
* if expiration is longer than recording's end time, then verify Episode1 is in COMPLETED\_STATE after its end time.
* Verify LeafRecordingRequets is in the DELETED\_STATE with DeletionDetail "EXPIRED" after the expiration period is up.

Test 64-67 TestSeriesDeleteRecording:TestDeleteSeriesParent-\*

* Set up a series recording with a Root ParentRecordingRequest(Show) which has a child branch that is also a ParentRecordingRequest (Season) which has a child that is a LeafRecordingRequest (Episode)
  + Test 64: Delete Root (Show) before Leaf (Episode's) recording startTime
  + Test 65: Delete Root (Show) after Leaf (Episode's) recording startTime
  + Test 66: Delete Branch (Season) before Leaf (Episode's) startTime
  + Test 67: Delete Branch(Season) after Leaf (Episode's) startTime

Test Verification:

* Verify the deleted ParentRecordingRequest and its children are no longer a member of in OcapRecordingManager's recordingList
* Verify an IllegalStateException is thrown when calling getKnownChildren()

on the deleted ParentRecordingRequest.

* Verify the leaf of the deleted ParentRecordingRequest shows "USER\_DELETED" as its DeletionDetails reason.

Test 68-73 TestSeriesDeleteRecording:TestDeleteSeriesLeaf

* Set up a series recording with a Root ParentRecordingRequest(Show) which has a child branch that is also a ParentRecordingRequest (Season) which has a child that is a LeafRecordingRequest (Episode)
* Delete the Episode (LeafRecordingRequest) either while it is

PENDING (71, 73, 75) or while it is IN\_PROGRESS (72, 74, 76)

* Test 68, 69: Root - PARTIALLY\_RESOLVED\_STATE

Branch - COMPLETELY\_RESOLVED\_STATE

* Test 70, 71: Root - PARTIALLY\_RESOLVED\_STATE

Branch - PARTIALLY\_RESOLVED\_STATE

* Test 72, 73: Root - COMPLETELY\_RESOLVED\_STATE

Branch - COMPLETELY\_RESOLVED\_STATE

Test Verification:

* Verify all ParentRecordingRequests (Show and Season) are still present
* Verify getKnownChildren() size: Root (Show) is 1 and Branch (Season) is 0
* Verify LeafRecordingRequest (Episode) is not part of RecordingList returned by the OcapRecordingManager
* Verify the leaf of the deleted ParentRecordingRequest shows "USER\_DELETED" as its DeletionDetails reason.

Test 74 TestSeriesReschedule:TestParentReschedule

* Set up a series recording with 2 leaf recordingRequests both are children of the same ParentRecordingRequest(Branch1) which is a child of the Root ParentRecordingRequest:

Root

|

Branch1

/ \

Leaf1 Leaf2

* Reschedule so that another ParentRecordingRequest is added as a child of Root ParentRecordingRequest; then add a child LeafRecordingRequest to the new child ParentRecordingRequest:

Root

| \

Branch1 Branch2

/ \ |

Leaf1 Leaf2 Leaf3

Test Verification:

* Test validates the children cout of Root, Branch1, and Branch2

Test 75-76 TestSeriesContention:TestSeriesContention1

* Schedules three sets of parallel series recordings
* Root1/Branch1/Leaf1 with Leaf1 to record from minute1 to minute2
* Root2/Branch2/Leaf2 with Leaf2 to record from minute1 to minute2
* Root3/Branch3/Leaf3 with Leaf3 to record from minute1 to minute2

Test Verification:

* After recording's endtime, Leaf1 and Leaf2 should bein the COMPLETED\_STATE while Leaf3 should be in the FAILED\_STATE.

Test 77-78 TestSeriesContention:TestSeriesContention\_delete\*

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedules three sets of series recordings:
  + Root3/Branch3/Leaf3 with Leaf3 to record from minute2 to minute4
  + Root1/Branch1/Leaf1 with Leaf1 to record from minute1 to minute3
  + Root2/Branch2/Leaf2 with Leaf2 to record from minute1 to minute3
* Test then deletes (Root1 by request for test 77; or Leaf1 by service for test 78).

Test Verification:

* Leaf1 and Leaf2 goes from PENDING\_NO\_CONFLICT\_STATE to IN\_PROGRESS\_STATE while Leaf3 stays in PENDING\_WITH\_CONFLICT\_STATE prior to deletion
* After deleteion, test verified Leaf1 goes to DELETED\_STATE while Leaf2 stays in IN\_PROGRESS\_STATE then moves to COMPLETED\_STATE; Leaf3 will also transition to PENDING\_NO\_CONFLICT\_STATE followed by IN\_PROGRESS and end in COMPLETED\_STATE.

Test 79 TestTunerConflict\_Gating:TestRecordingAndServiceConflict\_ABC

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedule 2 overlapping recordings ("#1-RecA" and "#2-RecB") to start while a serviceContext ("#3-SvcCtxtC") is presenting.

Test Verification:

* Verify Resource Contention Handler prioritized the two recordings over the serviceContext and the two recordings transitions to IN\_PROGRESS\_STATE successfully.
* Verify resource contention warning was only envoked once.

Test 80 TestTunerConflict\_Gating:TestRecordingAndServiceConflict\_ACB

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedule 2 overlapping recordings ("#1-RecA" and "#3-RecB") to start while a serviceContext ("#2-SvcCtxtC") is presenting.

Test Verification:

* Verify Resource Contention Handler prioritized "#1-RecA" and "#2-SvcCtxtC" over recording "#3-RecB" such that "#3-RecB" was never started and transitioned to FAILED\_STATE"
* Verify resource contention warning was only envoked once.

Test 81 TestTunerConflict\_Gating:TestRecordingAndServiceConflict\_CBA

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedule 2 overlapping recordings ("#3-RecA" and "#2-RecB") to start while a serviceContext ("#1-SvcCtxtC") is presenting.

Test Verification:

* Verify Resource Contention Handler prioritized "#2-RecB" and "#1-SvcCtxtC" over recording "#3-RecA" such that "#3-RecA" transitions from IN\_PROGRESS\_STATE to IN\_PROGRESS\_WITH\_ERROR state when "#2-RecB" starts
* Verify resource contention warning was only envoked once.

Test 82 TestTunerConflict\_Gating:TestRecordingNIAndSCConflict\_ABC

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedule a recordings ("#1-RecA") to start while both a ServiceContext ("#2-SvcCtxtB") and a NetworkInterface ("#3-niC") are presenting.

Test Verification:

* Verify Resource Contention Handler prioritized "#1-RecA" and "#2-SvcCtxtB" over NI presentation "#3-niC" such that "#1-RecA" transitions to IN\_PROGRESS\_STATE successfully while the Network Interface becomes unreserved.
* Verify resource contention warning was not envoked.

Test 83 TestTunerConflict\_Gating:TestRecordingNIAndSCConflict\_ABC

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedule a recordings ("#1-RecA") to start while both a ServiceContext ("#3-SvcCtxtB") and a NetworkInterface ("#2-niC") representing.

Test Verification:

* Verify Resource Contention Handler prioritized "#1-RecA" and "#2-niC" over "#3-svcCtxtB" such that "#1-RecA" transitions to IN\_PROGRESS\_STATE successfully while the Network Interface remains reserved.
* Verify resource contention warning was not envoked.

Test 84 TestTunerConflict\_Gating:TestRecordingNIAndSCConflict\_CBA

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedule a recordings ("#3-RecA") to start while both a ServiceContext ("#2-SvcCtxtB") and a NetworkInterface ("#1-niC") are presenting.

Test Verification:

* Verify Resource Contention Handler prioritized "#1-niC" and "#2-SvcCtxtB" over recording "#3-recA" such that "#1-RecA" transitions to FAILED\_STATE from the PENDING\_STATE while the Network Interface remains reserved.
* Verify resource contention warning was envoked once.

Test 85 TestTunerConflict\_Gating:TestRecordingAndServiceConflict\_ABCD

* Registers a default ResourceContentionHandler that prioritizes recordings by name lexigraphically
* Schedule three overlapping recordings ("#1-RecA", "#3-RecC", "#2-RecB") to start while a ServiceContext ("#4-SvcCtxtC") is presenting.

Test Verification:

* Verify Resource Contention Handler prioritized "#1-recA" over "#2-RecB" over "#3-recC" over "#4-svcCtxtC" such that "#1-RecA" and "#2-RecB" transitioned to IN\_PROGRESS\_STATE from the PENDING\_NO\_CONFLICT\_STATE while "#3-recC" transitioned to FAILED\_STATE from the PENDING\_WITH\_CONFLICT\_STATE remains reserved.
* Verify resource contention warning was invoked once.