

# **Joint Tactical Networking Center Test and Evaluation Laboratory**

## **Software Communications Architecture Version 2.2.2 Manual Operating Environment Software Test Description v3.3A**

### **Appendix B Volume 3 Test Procedures for Base Device Interfaces**

**13 April 2022**



**Prepared for:**

Joint Tactical Networking Center  
33000 Nixie Way, Bldg 50, Suite 339  
San Diego, CA 92147-5110

**Prepared by:**

Joint Tactical Networking Center Test and Evaluation Laboratory

## TABLE OF CONTENTS

<b>APPENDIX B VOLUME 3 TEST PROCEDURES.....</b>	<b>4</b>
B.3. Volume 3 Base Device Interfaces.....	4
B.3.1. OE_TC_004 - Device :: allocateCapacity.....	4
B.3.2. OE_TC_022 - AggregateDevice devices.....	10
B.3.3. OE_TC_023 - AggregateDevice :: addDevice.....	18
B.3.4. OE_TC_025 - AggregateDevice :: addDevice FAILURE_ALARM.....	27
B.3.5. OE_TC_027 - AggregateDevice :: addDevice raises InvalidObjectReference.....	33
B.3.6. OE_TC_028 - AggregateDevice :: removeDevice.....	40
B.3.7. OE_TC_029 - AggregateDevice :: removeDevice FAILURE_ALARM.....	46
B.3.8. OE_TC_030 - AggregateDevice :: removeDevice raises InvalidObjectReference.....	52
B.3.9. OE_TC_058 - Device operationalState.....	58
B.3.10. OE_TC_062 - Base Device Interfaces.....	70
B.3.11. OE_TC_079 - Device :: usageState.....	76
B.3.12. OE_TC_080 - Device :: adminState commanded to be LOCKED.....	89
B.3.13. OE_TC_081 - Device :: allocateCapacity.....	103
B.3.14. OE_TC_082 - Device :: allocateCapacity BUSY.....	112
B.3.15. OE_TC_083 - Device :: allocateCapacity Failure.....	120
B.3.16. OE_TC_084 - Device :: deallocateCapacity.....	126
B.3.17. OE_TC_085 - Device :: deallocateCapacity usageState.....	132
B.3.18. OE_TC_086 - Device :: deallocateCapacity raises InvalidCapacity.....	139
B.3.19. OE_TC_087 - Device :: releaseObject.....	145
B.3.20. OE_TC_088 - Device :: adminState attribute changes.....	152
B.3.21. OE_TC_089 - Device :: releaseObject raises ReleaseError exception.....	160
B.3.22. OE_TC_092 - Device :: Logical Device executable parameters.....	166
B.3.23. OE_TC_094 - Device :: Logical Device - CORBA.....	175
B.3.24. OE_TC_095 - Device :: Logical Device - CORBA Register.....	182
B.3.25. OE_TC_096 - Aggregate Device.....	189

---

B.3.26. OE_TC_097 - Device :: Logical Devices - CF Interfaces .....	197
B.3.27. OE_TC_098 - Device :: Logical Device allocation properties.....	205
B.3.28. OE_TC_118 - Device :: allocateCapacity's acceptable properties.....	216
B.3.29. OE_TC_130 - LoadableDevice :: load raises LoadFail .....	223
B.3.30. OE_TC_133 - LoadableDevice :: load raises InvalidFileName.....	229
B.3.31. OE_TC_134 - LoadableDevice :: unload raises InvalidFileName.....	235
B.3.32. OE_TC_135 - ExecutableDevice :: execute raises InvalidFileName .....	242
B.3.33. OE_TC_145 - ExecutableDevice :: terminate raises InvalidProcess.....	248
B.3.34. OE_TC_153 - LoadableDevice :: load.....	254
B.3.35. OE_TC_218 - Device Attributes .....	259
B.3.36. OE_TC_221 - ExecutableDevice Types.....	271
B.3.37. OE_TC_222 - ExecutableDevice :: execute raises exceptions.....	277
B.3.38. OE_TC_227 - ExecutableDevice :: execute .....	289
B.3.39. OE_TC_229 - Device :: allocateCapacity raises exceptions .....	298
B.3.40. OE_TC_230 - Device :: deallocateCapacity raises InvalidState .....	304
Index of Test Case Titles for Manual Tests.....	309

## APPENDIX B VOLUME 3 TEST PROCEDURES

### B.3. Volume 3 Base Device Interfaces

This volume consists of manual test cases, which verify requirements related to SCA Base Device Interfaces.

#### B.3.1. OE\_TC\_004 - Device :: allocateCapacity

**Test Case Number:** OE\_TC\_004

Device::allocateCapacity

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0407	The <i>allocateCapacity</i> operation shall set the <i>usageState</i> attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation (reference Figure 3-22).

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-62, Section 3.1.3.3.1.5.1.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page 28

#### Test Objective

This test case verifies OE0407. The test will verify that the *allocateCapacity* operation sets the *usageState* attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation.

#### Places to Verify

Devices

#### IDL References

#### Data

CF::Properties capacities;

#### Operations

boolean allocateCapacity ( in CF::Properties capacities )

raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);

## Preconditions

- All of the source code files with the Device *allocateCapacity* operations for the OE under test are available.

## Test Description

For the Device *allocateCapacity* operation within the OE under test:

- A. Identify the source code files that implement the *allocateCapacity* operation.
- B. Verify that the *allocateCapacity* operation sets the *usageState* attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation (OE0407).
  1. **Fail:** The *allocateCapacity* operation sets the *usageState* attribute to ACTIVE, when capacity is being used and no capacities are available for allocation.
  2. **Fail:** The *allocateCapacity* operation sets the *usageState* attribute to ACTIVE, when capacity is not being used and all the capacities are available for allocation.
  3. **Fail:** The *allocateCapacity* operation sets the *usageState* attribute to ACTIVE, when capacity is not being used and any of the capacities is available for allocation.
  4. **Pass:** The *allocateCapacity* operation sets the *usageState* attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_004				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the source code files directories that implement the <i>allocateCapacity</i> operation.</b>				
1. Locate and record the source code files that implement the <i>allocateCapacity</i> operation for the OE under test.	The files are recorded.  Discontinue test if directories are missing.		May need the help of the software engineer to locate the source code files directories.	
<b>B. Verify that the <i>allocateCapacity</i> operation sets the <i>usageState</i> attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation (OE0407).</b>				

OE_TC_004				
Steps	Expected Results	Actual Results	Comments	Test Result
2. Examine the source code files and verify that the <i>allocateCapacity</i> operation sets the <i>usageState</i> attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation.	<p><b>Fail:</b> The <i>allocateCapacity</i> operation sets the <i>usageState</i> attribute to ACTIVE, when capacity is being used and no capacities are available for allocation. (OE0407)</p> <p><b>Fail:</b> The <i>allocateCapacity</i> operation sets the <i>usageState</i> attribute to ACTIVE, when capacity is not being used and all the capacities are available for allocation. (OE0407)</p> <p><b>Fail:</b> The <i>allocateCapacity</i> operation sets the <i>usageState</i> attribute to ACTIVE, when capacity is not being used and any of the capacities is available for allocation. (OE0407)</p> <p><b>Pass:</b> The <i>allocateCapacity</i> operation sets the <i>usageState</i> attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation. (OE0407)</p>		<p>Sample IDL code:</p> <pre> boolean allocateCapacity (     in CF::Properties capacities )     raises     (CF::Device::InvalidCapacity,     CF::Device::InvalidState); </pre>	
<b>End of Test</b>				

Test Recording Log – OE_TC_004		
Step1 (Source Code Directories)	Step2 (Source Code Files)	Step3 (Failed Source Code Files)

### Test Summary OE\_TC\_004

Once testing is complete for every component of the OE under test, report the test result as follows:



**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0407 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_**Date Tested:** \_\_\_\_\_**Witness:** \_\_\_\_\_

### B.3.2. OE\_TC\_022 - AggregateDevice devices

#### Test Case Number: OE\_TC\_022

AggregateDevice devices

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0459	The readonly devices attribute shall contain a list of devices that have been added to this device or a sequence length of zero if the device has no aggregation relationships with other devices.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-74; Section 3.1.3.3.4.4.1 Page 3-93; Section 3.1.3.6.2
SCA Appendix C: Core Framework IDL	FINAL/ 15 May 2006, V2.2.2	Page C-5

#### Test Objective

This test case verifies OE0459. The objective of this test is to verify that the readonly devices attribute will contain a list of devices that have been added to this device or a sequence length of zero if the device has no aggregation relationships with other devices.

#### Places to Verify

Aggregate Devices

#### IDL References

##### Data

```
typedef sequence <Device> DeviceSequence;  
readonly attribute CF::DeviceSequence devices;  
readonly attribute CF::DeviceSequence registeredDevices;
```

##### Operations

```
void addDevice (in CF::Device associatedDevice) raises (CF::InvalidObjectReference);
```

## Preconditions

- Domain Profile test has passed
- All the Domain Profile files are available
- The source code and XML files for the OE are available.

## Test Description

- A. Determine the devices, if any, (the number of, names and location of the devices) of the Operating Environment from the DCD file(s) of the Domain Profile and/or developer documentation. (OE0459)
1. **Pass:** The DCD file(s) exist and indicates the name(s) of the devices.
  2. **Fail:** If there are devices and there are no DCD files.

For every device, perform the rest of the test case steps

- B. Identify all aggregate and non-aggregate devices. (OE0459)
1. **Pass:** The logic is found in the source code that handles devices and any other devices with an aggregate relationship.
  2. **Pass:** The logic is found in the source code that handles devices which does not have an aggregate relationship.
  3. **Untested:** The logic is NOT found in the source code that handles devices and any other devices with an aggregate relationship.
  4. **Untested:** The logic is NOT found in the source code that handles devices and any other devices without an aggregate relationship.
- C. Verify that the devices attribute exist for a parent device in an aggregated relationship and that it is the correct type. (OE0459).
1. **Pass:** There is an attribute whose name is devices and its type is DeviceSequence.
  2. **Fail:** There is not an attribute whose name is devices and its type is DeviceSequence.
- D. Verify in the code for the AggregateDevice that the readonly devices attribute contains a list of devices that have been added to this device. (OE0459).
1. **Pass:** This parent device has a devices attribute, which contains a list of child devices that have been added to the attribute list.
  2. **Fail:** This parent device does not have a devices attribute, which contains a list of child devices that have been added to the attribute list.
- E. Verify that the devices attribute has a sequence length of zero if the device has no aggregation relationship with other devices. (OE0459).
1. **Pass:** Code is located where the device attribute has a sequence length of zero if the device has no aggregation relationship with other devices.
  2. **Fail:** Code is NOT located where the device attribute has a sequence length of zero if the device has no aggregation relationship with other devices.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_022				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine the devices, if any, (the number of, names and location of the devices) of the Operating Environment from the DCD file(s) of the Domain Profile and/or developer documentation. (OE0459)</b>				
1. Locate the DCD file(s) and record the name of each device.	<p><b>Pass:</b> The DCD file(s) exist(s) and indicate(s) the name(s) of the devices. (OE0459)</p> <p><b>Fail:</b> If there are devices and there are no DCD files. (OE0459)</p>		<p><b>NOTE:</b> The fastest way to determine the names of the devices is to consult with the developer and OE documentation.</p> <p>A device's id and name are usually specified in the DCD file which could be similar to the following example:</p> <pre>&lt;componentplacement&gt;   &lt;componentfileref refid="xxxxxx"/&gt;   &lt;componentinstantiation     id="DCE:ddd123xxxx"&gt;     &lt;usagename&gt;DeviceName&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt;</pre> <pre>&lt;componentfiles&gt; &lt;componentfile id="DCE:ddddxxxx " type="SPD"&gt;   &lt;localfile name=".../logservice/DeviceFile.spd.xml"/ &gt;   &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre>	

OE_TC_022				
Steps	Expected Results	Actual Results	Comments	Test Result
2. Examine the DCD file and locate all occurrences of SPD declarations. Record the SPD file(s).	<b>Pass:</b> One or more SPD file exists. (OE0459)  <b>Fail:</b> A SPD file does not exist. (OE0459)		<pre> &lt;componentfiles&gt;   &lt;componentfile id="DCE:ddddxxxx "     type="SPD"&gt;     &lt;localfile       name=".../xxxxx.spd.xml"/&gt;     &lt;/componentfile&gt;   &lt;/componentfiles&gt; </pre>	
3. Locate the source code for the device and record the name of the file.	<b>Pass:</b> The source code of the devices is located. (OE0459)  <b>Untested:</b> The source code of the devices is not located. (OE0459)		<p>The development engineer and the developer documentation, if available, is the best source for this information.</p> <p>An executable file may be identified in the SPD file and the related source code file.</p> <p>Executable declaration may look similar to:</p> <pre> &lt;implementation id = "DCE:nnn"&gt;   &lt;code type="Executable"&gt;     &lt;localfile name="mmm.exe"/&gt;   &lt;/code&gt; &lt;/implementation&gt; </pre>	
<b>For every device, perform the rest of this test case steps,</b>  <b>B. Identify all aggregate and non-aggregate devices. (OE0459)</b>				

OE_TC_022				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Look at the source code for the devices having aggregate relationships and verify that there is logic related to making this device part of an aggregate relationship.  (The next step addresses the case when a device has no aggregation relationships.)	<b>Pass:</b> There is logic to add this device as a child device to a parent device. (OE0459)  <b>Fail:</b> There is no logic to add this device as a child device to a parent device. (OE0459)		A device is not required to have an aggregate relationship with any other device.	
5. Look at the source code for the devices not having aggregate relationships and verify that there is no logic related to making this device part of an aggregate relationship. This info will be needed for an upcoming step (in section E).	<b>Pass:</b> There is no logic to add this device as a child device to a parent device. (OE0459)  <b>Fail:</b> There is logic to add this device as a child device to a parent device. (OE0459)		A device is not required to have an aggregate relationship with any other device.	
<b>C. Verify that the devices attribute exist for a parent device in an aggregated relationship and that it is the correct type. (OE0459)</b>				
6. Verify that there is an attribute whose name is devices and its type is DeviceSequence, which would contain a list of devices. (Hint: look for the word 'sequence'.)	<b>Pass:</b> There is an attribute whose name is devices and its type is DeviceSequence. (OE0459)  <b>Fail:</b> There is not an attribute whose name is devices and its type is DeviceSequence. (OE0459)		This is from Section 3.1.3.3.4.4.1, page 3-74 of the SCA v2.2.2 readonly attribute CF::DeviceSequence devices;	
<b>D. Verify that the readonly devices attribute contains a list of child devices that have been added to this parent device. (OE0459)</b>				

OE_TC_022				
Steps	Expected Results	Actual Results	Comments	Test Result
7. Verify in the code for the AggregateDevice that the readonly devices attribute contains a list of devices that have been added to this device.	<p><b>Pass:</b> This parent device has a devices attribute, which contains a list of child devices that have been added to the attribute list. (OE0459)</p> <p><b>Fail:</b> This parent device does not have a devices attribute, which contains a list of child devices that have been added to the attribute list. (OE0459)</p>			
<b>E. Verify that the readonly devices attribute has a sequence length of zero if the device has no aggregation relationship with other devices. (OE0459)</b>				
8. Verify that the devices attribute has a sequence length of zero if the device has no aggregation relationship with other devices.	<p><b>Pass:</b> Code is located where the devices attribute has a sequence length of zero if the device has no aggregation relationship with other devices. (OE0459)</p> <p><b>Fail:</b> Code is NOT located where the device attribute has a sequence length of zero if the device has no aggregation relationship with other devices. (OE0459)</p>			
<b>End of Test</b>				

Test Recording Log – OE_TC_022					
DCD file:					
SPD file(s)					
Step 3 (devices and aggregate devices)	Step 4 (aggregate relationship)	Step 5 (no aggregate relationship)	Step 6 (DeviceSequence)	Step 7 (list of devices)	Step 8 (length of zero)



## Test Summary OE\_TC\_022

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0459 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.3. OE\_TC\_023 - AggregateDevice :: addDevice

**Test Case Number:** OE\_TC\_023

**Device::**AggregateDevice::addDevice

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0460	The <i>addDevice</i> operation shall add the input associatedDevice parameter to the <i>AggregateDevice</i> 's devices attribute when the associatedDevice does not exist in the devices attribute.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-75, Section 3.1.3.3.4.5.1.3
SCA Appendix C: Core Framework IDL	FINAL/ 15 May 2006, V2.2.2	Page C-5

#### Test Objective

This test case verifies OE0460. The objective of this test is to verify that the *addDevice* operation will add the input *associatedDevice* parameter to the *AggregateDevice*'s devices attribute when the *associatedDevice* does not exist in the devices attribute. The purpose of the addDevice is to provide the mechanism to associate a device with another device.

#### Places to Verify

Aggregate Devices

#### IDL References

##### Exceptions

exception InvalidObjectReference {string msg;};

##### Operations

void addDevice (in Device associatedDevice) raises  
(InvalidObjectReference)

##### Preconditions

- Domain Profile test has passed
- All the Domain Profile files are available

- The source code and XML files for the OE are available.

## Test Description

A. Determine the devices, if any, of the Operating Environment from the DCD file(s) of the Domain Profile. (OE0460)

1. **Pass:** The DCD file(s) exist which indicates the name(s) of the devices.
2. **Fail:** If there are devices and there are no DCD files.

For every device, perform the rest of the test case steps:

B. Identify all aggregate devices.

1. Verify that the *AggregateDevice* interface exists. (OE0460)
  - a. **Pass:** The *AggregateDevice* interface exists
  - b. **Fail:** The *AggregateDevice* interface does not exist.
2. Verify that the *addDevice* operation exists. (OE0460)
  - a. **Pass:** The *addDevice* operation exists.
  - b. **Fail:** The *addDevice* operation does not exist.
3. Verify that the *addDevice* operation adds the input *associatedDevice* parameter to the *AggregateDevice*'s devices attribute when the *associatedDevice* does not exist in the devices attribute (OE0460) and the *associatedDevice* is not a nil CORBA object reference. (OE0460)
  - a. **Pass:** The *addDevice* operation adds the input *associatedDevice* parameter to the *AggregateDevice*'s devices attribute when the *associatedDevice* does not exist in the devices attribute and the *associatedDevice* is not a nil CORBA object reference.
  - b. **Fail:** The *addDevice* operation does not add the input *associatedDevice* parameter to the *AggregateDevice*'s devices attribute when the *associatedDevice* does not exist in the devices attribute.
  - c. **Fail:** There is no check in the source code to check for a nil object.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_023				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine the devices, if any, of the Operating Environment from the DCD file(s) of the Domain Profile. (OE0460)</b>				
1. Locate the DCD file(s) and record the name of each device.	<p><b>Pass:</b> The DCD file(s) exist(s) which indicate(s) the name(s) of the devices. (OE0460)</p> <p><b>Fail:</b> If there are devices and there are no DCD files. (OE0460)</p>		<p><b>NOTE 1:</b> The fastest way to determine the names of the devices is to consult with the developer and OE documentation.</p> <p><b>NOTE 2:</b> A single DCD file may include references to multiple devices.</p> <p>A device's id and name are usually specified in the DCD file which could be similar to the following example:</p> <pre>&lt;componentplacement&gt;   &lt;componentfileref refid="xxxxxx"/&gt;   &lt;componentinstantiation     id="DCE:ddd123xxxx"&gt;     &lt;usagename&gt;DeviceName&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt; &lt;componentfiles&gt;   &lt;componentfile id="DCE:ddddxxxx "     type="SPD"&gt;     &lt;localfile       name=".../logservice/DeviceFile.spd.xml"/&gt;     &lt;/componentfile&gt;   &lt;/componentfiles&gt;</pre>	
2. Examine the DCD file(s) and locate all occurrences of SPD declarations. Record the SPD file(s) associated with each device.	<p><b>Pass:</b> One or more SPD file exists. (OE0460)</p> <p><b>Fail:</b> A SPD file does not exist for the device. (OE0460)</p>		<pre>&lt;componentfiles&gt;   &lt;componentfile id="DCE:ddddxxxx "     type="SPD"&gt;     &lt;localfile       name=".../xxxxx.spd.xml"/&gt;     &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre>	

OE_TC_023				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Locate the source code for the device and record the name of the file.	<p><b>Pass:</b> The source code of the devices is located. (OE0460)</p> <p><b>Untested:</b> The source code of the devices is not located. (OE0460)</p>		<p>The development engineer and the developer documentation, if available, are the best source for this information.</p> <p>An executable file may be identified in the SPD file and the related source code file.</p> <p>Executable declaration may look similar to:            &lt;implementation id = "DCE:nnn"&gt;              &lt;code type="Executable"&gt;                &lt;localfile name="mmm.exe"/&gt;              &lt;/code&gt;            &lt;/implementation&gt;</p>	
<p><b>For every device, perform the rest of the test case steps,</b></p> <p><b>B. Identify all aggregate devices. (OE0460)</b></p>				
4. Look at the source code for the devices having aggregate relationships and verify that there is logic related to making this device part of an aggregate relationship. (Repeat loop for next device, if this device is not an aggregate device.)	<p><b>Pass:</b> There is logic to add this device as a child device to a parent device. (OE0460)</p> <p><b>Fail:</b> There is no logic to add this device as a child device to a parent device. (OE0460)</p>		<p>A device is not required to have an aggregate relationship with any other device.</p> <p>May need the help of the software engineer.</p>	
<b>1. Verify that the <i>AggregateDevice</i> interface exists. (OE0460)</b>				

OE_TC_023				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Examine the source code files and verify that the <i>AggregateDevice</i> interface exists.	<b>Pass:</b> <i>AggregateDevice</i> interface exists. (OE0460)  <b>Fail:</b> <i>AggregateDevice</i> interface does not exist. (OE0460)		Sample code:  <pre>interface AggregateDevice { readonly attribute CF::DeviceSequence devices;     void addDevice ( in CF::Device associatedDevice) raises (CF::InvalidObjectReference);     void removeDevice ( in CF::Device associatedDevice) raises (CF::InvalidObjectReference); };</pre>	
<b>2. Verify that the <i>addDevice</i> operation exists. (OE0460)</b>				
6. Examine the source code files and verify that the <i>addDevice</i> operation exists.	<b>Pass:</b> The <i>addDevice</i> operation exists. (OE0460)  <b>Fail:</b> The <i>addDevice</i> operation does not exist. (OE0460)		Sample code:  <pre>void addDevice ( in CF::Device     as associatedDevice) raises (CF::InvalidObjectReference);</pre>	
<b>3. Verify that the <i>addDevice</i> operation adds the input <i>associatedDevice</i> parameter to the <i>AggregateDevice</i> 's <i>devices</i> attribute when the <i>associatedDevice</i> does not exist in the <i>devices</i> attribute (OE0460) and the <i>associatedDevice</i> is not a nil CORBA object reference. (OE0460)</b>				

OE_TC_023				
Steps	Expected Results	Actual Results	Comments	Test Result
7. Examine the source code files and verify that there is an existence of some method/function that compares the input <i>associatedDevice</i> parameter with the device's device attribute. (Note: for the Pass condition, if the method/function for the <i>associatedDevice</i> does exist in the device's devices attribute, then no need to do the next steps, #8 and 9.)	<p><b>Pass:</b> An examination of the source code files verifies the existence of some method/function that compares the <i>associatedDevice</i> parameter with the device's device attribute. (OE0460)</p> <p><b>Fail:</b> An examination of the source code files verifies the lack of an existence of some method/function that compares the <i>associatedDevice</i> parameter with the device's device attribute. (OE0460)</p>		<p>Tests the operation on the device's attribute for the input's existence within it.</p> <p>From the SCA v2.2.2 Appendix C, IDL document, the following shows the attributes for devices: readonly attribute CF::DeviceSequence devices;</p>	
8. Examine the source code to determine if there is an 'if' condition check if the <i>associatedDevice</i> parameter is a nil CORBA object reference.	<p><b>Pass:</b> There is a check in the source code to prevent adding a nil object into the device's devices attribute. (OE0460)</p> <p><b>Fail:</b> There is no check in the source code to prevent adding a nil object into the device's devices attribute. (OE0460)</p>		<p>See test case OE_TC_027 for OE0462, which refers to the nil condition for raising an <i>InvalidObjectReference</i> exception. (Note: this covers the gray area where a nil object could be added to the <i>AggregateDevice</i>'s devices attribute. So, the <i>associatedDevice</i> parameter is there, but it is nil.)</p>	

OE_TC_023				
Steps	Expected Results	Actual Results	Comments	Test Result
9. Examine the source code files to verify that the <i>addDevice</i> operation adds the input <i>associatedDevice</i> parameter to the <i>AggregateDevice</i> 's devices attribute when the <i>associatedDevice</i> does not exist in the devices attribute.	<b>Pass:</b> The <i>addDevice</i> operation adds the input <i>associatedDevice</i> parameter. (OE0460)  <b>Fail:</b> The <i>addDevice</i> operation does not add the input <i>associatedDevice</i> parameter. (OE0460)  <b>Fail:</b> Should not proceed if there is a non-existent object in the device's devices attribute (i.e., null/nil). (OE0460)			
End of Test				



Test Recording Log – OE_TC_023							
Step 1 (DCD)	Step 2 (SPD)	Step 3 (Source Code)	Step 4 (make aggregate relationship)	Step 5 (AggregateDe vice interface)	Step 6 (addDevice)	Step 7 (is/is not)	Step 8 (adds if needed)

## Test Summary OE\_TC\_023

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s) (x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0460 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.4. OE\_TC\_025 - AggregateDevice :: addDevice FAILURE\_ALARM

**Test Case Number:** OE\_TC\_025

AggregateDevice::addDevice

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0461	The <i>addDevice</i> operation shall write a FAILURE_ALARM log record, upon unsuccessful adding of an <i>associatedDevice</i> to the <i>AggregateDevice's devices</i> attribute.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-74 and 3-75, Section 3.1.3.3.4.5.1.3

#### Test Objective

This test case verifies OE0461. The objective of this test case is to verify that the *addDevice* operation writes a FAILURE\_ALARM log record when an *associatedDevice* cannot be successfully added to the *AggregateDevice's devices* attribute.

#### Places to Verify

AggregateDevice

#### IDL References

```
interface AggregateDevice {  
    void addDevice ( in CF::Device associatedDevice )  
        raises (CF::InvalidObjectReference);}
```

#### Preconditions

- Source code files with the *addDevice* operation for the OE under test are available.

#### Test Description

For each AggregateDevice within the OE under test:

A. Verify that the *AggregateDevice* interface exists. (OE0461)

1. **Pass:** The *AggregateDevice* interface exists.
  2. **Untested:** The *AggregateDevice* interface does not exist.
- B. Verify that the *addDevice* operation exists in the source code. (OE0461)
1. **Pass:** The *addDevice* operation exists.
  2. **Fail:** The *addDevice* operation does not exist.
- C. Verify that the *addDevice* operation writes a FAILURE\_ALARM log record when an associatedDevice cannot be successfully added to the *AggregateDevice's devices* attribute (OE0461).
1. **Pass:** The *addDevice* operation writes a FAILURE\_ALARM log record when an associatedDevice cannot be successfully added to the *AggregateDevice's devices* attribute
  2. **Fail:** The *addDevice* operation does not write a FAILURE\_ALARM log record when an associatedDevice cannot be successfully added to the *AggregateDevice's devices* attribute

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_025				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>For each AggregateDevice within the OE under test:</b>				
<b>A. Verify that the <i>AggregateDevice</i> interface exists.(OE0461)</b>				
1. Examine the source code files and verify that the <i>AggregateDevice</i> interface exists.	<b>Pass:</b> <i>AggregateDevice</i> interface exists.(OE0461)  <b>Untested:</b> <i>AggregateDevice</i> interface does not exist.(OE0461)		<pre>interface AggregateDevice { }</pre>	
<b>B. Verify that the <i>addDevice</i> operation exists in the source code. (OE0461)</b>				
2. Examine the source code files and verify that the <i>addDevice</i> operation exists.	<b>Pass:</b> The <i>addDevice</i> operation exists.(OE0461)  <b>Fail:</b> The <i>addDevice</i> operation does not exist. (OE0461)		<pre>void addDevice(     in CF::Device         associatedDevice)     raises     (CF::InvalidObjectReference);</pre>	
<b>C. Verify that the <i>addDevice</i> operation writes a FAILURE_ALARM log record when an associatedDevice cannot be successfully added to the <i>AggregateDevice's</i> devices attribute (OE0461).</b>				

OE_TC_025				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Examine the source code files and verify that the <i>addDevice</i> operation writes a FAILURE_ALARM log record when an associatedDevice cannot be successfully added to the <i>AggregateDevice's devices</i> attribute.	<b>Pass:</b> The <i>addDevice</i> operation writes a FAILURE_ALARM log record when an associatedDevice cannot be successfully added to the <i>AggregateDevice's devices</i> attribute. (OE0461)  <b>Fail:</b> The <i>addDevice</i> operation does not write a FAILURE_ALARM log record when an associatedDevice cannot be successfully added to the <i>AggregateDevice's devices</i> attribute. (OE0461)			
End of Test				

Test Recording Log –OE_TC_025		
Step1 (AggregateDevice interface exists)	Step2 (addDevice operation exists)	Step3 (FAILURE_ALARM record written)

## Test Summary OE\_TC\_025

Once testing is complete for every component of the OE under test, report the test result as follows:

<b>Pass:</b>	No failures detected	
<b>Fail:</b>	Failure(s) detected in Step(s)(x)	Failure of any associated criteria results in a failure of a requirement.
<b>Untested:</b>	Condition which is not testable	
<b>N/A:</b>	Not Applicable	

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0461\_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_**Date Tested:** \_\_\_\_\_**Witness:** \_\_\_\_\_



### B.3.5. OE\_TC\_027 - AggregateDevice :: addDevice raises InvalidObjectReference

#### Test Case Number: OE\_TC\_027

AggregateDevice::addDevice raises InvalidObjectReference

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0462	The <i>addDevice</i> operation shall raise the CF <i>InvalidObjectReference</i> when the input <i>associatedDevice</i> parameter is a nil CORBA object reference.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-75, Section 3.1.3.3.4.5.1.5 Page 3-93, Section 3.1.3.6.5
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Pages D-49 to D-51, Sections D.4.1.4.1 to D.7.1.4.1.5

#### Test Objective

This test case verifies OE0462. The objective of this test case is to verify that the *addDevice* operation raises the CF *InvalidObjectReference* when the input *associatedDevice* parameter is a nil CORBA object reference.

#### Places to Verify

AggregateDevice

#### IDL References

##### Exceptions

```
exception InvalidObjectReference { string msg;};
```

##### Operations

```
interface AggregateDevice
void addDevice (
    in CF::Device associatedDevice )
    raises (CF::InvalidObjectReference);
```

## Preconditions

- System design documents as they apply to aggregate devices.
- Source code files for the devices that implement the *AggregateDevice* interface are available.

## Test Description

A. Verify what devices implement the *AggregateDevice* interface. (OE0462)

1. **N/A:** There are no devices that implement the *AggregateDevice* interface.
2. **Pass:** There are device that implement the *AggregateDevice* interface.

For each Device which implements the *AggregateDevice* interface

B. Verify that the *AggregateDevice* interface exists. (OE0462)

1. **Fail:** The *AggregateDevice* interface does not exist.
2. **Pass:** The *AggregateDevice* interface exists.

C. Verify that the *addDevice* operation to add an *associatedDevice* to the *AggregateDevice*'s devices attribute exists. (OE0462)

1. **Fail:** The *addDevice* operation does not exist.
2. **Pass:** The *addDevice* operation exists.

D. Verify that the *addDevice* operation raises the CF *InvalidObjectReference* exception when the input *associatedDevice* parameter is a nil CORBA object reference. Confirm that some message is included in the raised exception. (OE0462).

1. **Fail:** The *addDevice* operation does not raise the CF *InvalidObjectReference* exception when the input *associatedDevice* parameter is a nil CORBA object reference.
2. **Pass:** The *addDevice* operation raises the CF *InvalidObjectReference* exception with a message when the input *associatedDevice* parameter is a nil CORBA object reference.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_027				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Verify what devices implement the <i>AggregateDevice</i> interface. (OE0462)</b>				
1. Obtain and record the directory where the Domain Profile (XML) files are located.	<b>Pass:</b> The directory is located. (OE0462)  <b>Untested:</b> The directory is not located. (OE0462)			
2. Find and identify the Device Configuration Descriptor(DCD) files. There will be one for each Device Manager.	These are optional files but there must be one for every Device Manager that exists. The file name format is like this, xxx.dcd.xml  <b>N/A:</b> The DCD files are not found. (OE0462)  <b>Pass:</b> The DCD files are found. (OE0462)			
3. Examine the DCD files and verify that there are lines containing the word <i>compositepartofdevice</i> .	<b>N/A:</b> The <i>compositepartofdevice</i> section is not in any DCD file. (OE0462)  <b>Pass:</b> At least one <i>compositepartofdevice</i> section is in the DCD files. (OE0462)		No <i>compositepartofdevice</i> entry means there are no AggregateDevices, because this component identifies a devices parent device.	
4. Find and record the devices listed as <i>compositepartofdevice</i> within the DCD file.	<b>N/A:</b> There are no devices listed as <i>compositepartofdevice</i> within the DCD files. (OE0462)  <b>Pass:</b> There are devices listed as <i>compositepartofdevice</i> within the DCD files. (OE0462)			

OE_TC_027				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>For each device which implements the AggregateDevice interface:</b> <b>B. Verify that the <i>AggregateDevice</i> interface exists. (OE0462)</b>				
5. Examine the source code files of the parent device and verify that the <i>AggregateDevice</i> interface exists.	<b>N/A:</b> The <i>AggregateDevice</i> interface does not exist. <b>This ends the test.</b> (OE0462)  <b>Pass:</b> The <i>AggregateDevice</i> interface exists. (OE0462)		Sample code:  CF:: <i>AggregateDevice</i> ...	
<b>C. Verify that the <i>addDevice</i> operation to add an associatedDevice to the <i>AggregateDevice</i>'s devices attribute exists. (OE0462)</b>				
6. Examine the source code files and verify that the <i>addDevice</i> operation to add an associatedDevice to the <i>AggregateDevice</i> 's devices attribute exists.	<b>Pass:</b> The <i>addDevice</i> operation exists. (OE0462)  <b>Fail:</b> The <i>addDevice</i> operation does not exist. (OE0462)		The following sample code is merely an example of what the code might look like. This is not to be construed as a measure of passing or failing.  <pre>void AggDev::addDevice(     CF::Device_ptr         associatedDevice)     raises         (CF::InvalidObjectReference);</pre>	
<b>D. Verify that the <i>addDevice</i> operation raises the CF <i>InvalidObjectReference</i> when the input <i>associatedDevice</i> parameter is a nil CORBA object reference. (OE0462).</b>				
7. Examine the source code files and verify that the <i>addDevice</i> operation raises the CF <i>InvalidObjectReference</i> when the input <i>associatedDevice</i> parameter is a nil CORBA object reference.	<b>Pass:</b> The <i>addDevice</i> raises the CF <i>InvalidObjectReference</i> when the input <i>associatedDevice</i> parameter is a nil CORBA object reference. (OE0462)  <b>Fail:</b> The <i>addDevice</i> operation does not raise the CF <i>InvalidObjectReference</i> when the input <i>associatedDevice</i> parameter		Somewhere in the <i>addDevice</i> implementation there should be a test (an if statement perhaps) for valid input parameters. For this requirement, a test for nil against the <i>associatedDevice</i> should result in raising the CF <i>InvalidObjectReference</i> exception if true.	

OE_TC_027				
Steps	Expected Results	Actual Results	Comments	Test Result
	is a nil CORBA object reference. (OE0462)			
8. Verify that a message is included in the <i>CF::InvalidObjectReference</i> exception upon it being raised	<b>Pass:</b> The raised <i>CF::InvalidObjectReference</i> exception contains a message string parameter. (OE0462)  <b>Fail:</b> The raised <i>CF::InvalidObjectReference</i> exception has no message string parameter. (OE0462)			
End of Test				

Test Recording Log – OE_TC_027						
Step 1 (Directory of the Domain Profile files)	Steps 2 & 3 (Found DCD file listing the parent devices) Y/N	Step 4 (List parent devices)	Step 5 (Aggregate Device interface exists) Y/N	Step 6 (addDevice operation exists) Y/N	Step 7 (raises exception if a nil CORBA object) Y/N	Step 8 (Message string is in exception) Y/N

## Test Summary OE\_TC\_027

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0462\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.6. OE\_TC\_028 - AggregateDevice :: removeDevice

**Test Case Number:** OE\_TC\_028

AggregateDevice::removeDevice

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0463	The <i>removeDevice</i> operation shall remove the input <i>associatedDevice</i> parameter from the <i>AggregateDevice</i> 's <i>devices</i> attribute.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 2-2, page 3-75, Section 3.1.3.3.4.5.2.3, page 3-96, Section 3.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	C-5

#### Test Objective

This test case verifies OE0463. The objective of this test is to verify that the *removeDevice* operation provides the mechanism to disassociate a device from another device. Using the input parameter *associatedDevice*, the *removeDevice* operation will remove devices from the *AggregateDevice device* attribute.

#### Places to Verify

Aggregate Devices

#### IDL References

##### Operations

void removeDevice ( in CF::Device associatedDevice ) raises (CF::InvalidObjectReference);

#### Preconditions

- Source code files with the *removeDevice* operation for the OE under test is available



## Test Description

For each *AggregateDevice* within the OE under test:

- A. Verify that the *AggregateDevice* interface exists. (OE0463)
  - 1. **Pass:** The *AggregateDevice* interface exists.
  - 2. **Untested:** The *AggregateDevice* interface does not exist.
- B. Verify that the *removeDevice* operation can be found in the source code. (OE0463)
  - 1. **Pass:** The *removeDevice* operation is found in the source code.
  - 2. **Fail:** The *removeDevice* operation is not found in the source code.
- C. Verify that the *removeDevice* operation removes the input *associatedDevice* parameter from the *AggregateDevice*'s *devices* attribute. (OE0463)
  - 1. **Pass:** The *removeDevice* operation removes the input *associatedDevice* parameter from the *AggregateDevice*'s *devices* attribute.
  - 2. **Fail:** The *removeDevice* operation does not remove the input *associatedDevice* parameter from the *AggregateDevice*'s *devices* attribute.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_028				
Steps	Expected Results	Actual Results	Comments	Test Result
For each AggregateDevice within the OE under test:				
<b>A. Verify that the <i>AggregateDevice</i> interface exists. (OE0463)</b>				
1. Examine the source code files and verify that the <i>AggregateDevice</i> interface exists.	<b>Pass:</b> The <i>AggregateDevice</i> interface exists. (OE0463)  <b>Untested:</b> The <i>AggregateDevice</i> interface does not exist. (OE0463)		May need the help of a software developer to locate the source code files.  <pre>interface AggregateDevice {</pre>	
<b>B. Verify that the <i>removeDevice</i> operation can be found in the source code. (OE0463)</b>				
2. Examine the source code files and verify that the <i>removeDevice</i> operation exists.	<b>Pass:</b> The <i>removeDevice</i> operation exists. (OE0463)  <b>Fail:</b> The <i>removeDevice</i> operation does not exist. (OE0463)		<pre>void removeDevice ( in CF::Device associatedDevice ) raises (CF::InvalidObjectReference);</pre>	
<b>C. Verify that the <i>removeDevice</i> operation removes the input <i>associatedDevice</i> parameter from the <i>AggregateDevice</i>'s devices attribute. (OE0463)</b>				

OE_TC_028				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Examine the source code files and verify that the <i>removeDevice</i> operation removes the input <i>associatedDevice</i> parameter from the <i>AggregateDevice</i> 's <i>device</i> attribute.	<b>Pass:</b> The <i>removeDevice</i> operation removes the input <i>associatedDevice</i> parameter from the <i>AggregateDevice</i> 's <i>device</i> attribute. (OE0463)  <b>Fail:</b> The <i>removeDevice</i> operation does not remove the input <i>associatedDevice</i> parameter from the <i>AggregateDevice</i> 's <i>device</i> attribute. (OE0463)		<i>void removeDevice( in CF::Device associatedDevice ) raises (CF::InvalidObjectReference);</i>	
<b>End of Test</b>				

Test Recording Log – OE_TC_028		
Step1 (AggregateDevice exists)	Step2 (removeDevice operation exists)	Step3 (associateDevice removed)

## Test Summary OE\_TC\_028

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0463 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.7. OE\_TC\_029 - AggregateDevice :: removeDevice FAILURE\_ALARM

**Test Case Number:** OE\_TC\_029

AggregateDevice::removeDevice

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0464	The <i>removeDevice</i> operation shall write a FAILURE_ALARM log record, upon unsuccessful removal of the associatedDevice from the AggregateDevice devices attribute.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pg. 3-7, Section 3.1.3.3.4.5.2.3

#### Test Objective

This test case verifies requirement OE0464. The AggregateDevice functionality permits child device(s) to be associated with a parent device (aggregate) and the *removeDevice()* operation allows for the disassembly of the aggregate. The objective of this test is to verify that an AggregateDevice can be properly disassociated, that failure during a *removeDevice()* operation results in the writing of a FAILURE\_ALARM log record.

#### Places to Verify

Aggregated Devices

#### IDL References

```
void removeDevice ( in CF::Device associatedDevice )  
    raises (CF::InvalidObjectReference);
```

#### Preconditions

- The source code files are available.

#### Test Description

A. Identify all aggregate devices. (OE0464)

1. **N/A:** No aggregate devices.

For each aggregate device perform the following steps:

- B. Determine the association of parent and children for aggregated Device.
- C. Verify the *removeDevice* method writes a FAILURE\_ALARM log record if the specified Device is not successfully removed. (OE0464)
  1. **Pass:** The FAILURE\_ALARM log record is written.
  2. **Fail:** The FAILURE\_ALARM log record is not written.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_029				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify all aggregate devices. (OE0464)</b>				
1. Determine the association of parent and child for aggregated Device.	The system under test may or may not have aggregate devices.  The requirement is not applicable (N/A) if there are no aggregate device implementations. (OE0464)		In an aggregate (system) of devices there may be prescribed restrictions or orders that restrict how to perform the association and disassociation of child devices.	
<b>For each aggregate device perform the following steps</b>				
<b>B. Determine the association of parent and children for aggregated Device. (OE0464)</b>				
2. Locate the source code file for the parent device. Locate the code for the <i>removeDevice</i> operation, and locate the code for the <i>addDevice</i> operation.	Source code located. Record the filename and line number in the test-recording log.			
3. Review code comments in both operations and device-specific documentation of the parent and child devices.	Device-specific instructions for the parent device and each of the child devices.		In an aggregate (system) of devices there may be prescribed restrictions or orders that restrict how to perform the association and disassociation of child devices.	
<b>C. Verify the <i>removeDevice</i> method writes a FAILURE_ALARM log record (OE0464)</b>				



OE_TC_029				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Verify the <i>removeDevice</i> method writes a FAILURE_ALARM log record if the specified Device is not successfully removed.	<b>Pass:</b> The FAILURE_ALARM log record is written. (OE0464)  <b>Fail:</b> The FAILURE_ALARM log record is not written. (OE0464)		The SCA does not give specifics for this particular FAILURE_ALARM log record, but elsewhere the expected format of a similar record is "installApplication::invalid file is xxx", where xxx is the input or referenced filename.	
<b>End of Test</b>				

Test Recording Log – OE_TC_029				
Step1 Name of the (parent) Aggregate Device	Step2 () Source Code File Name	Step2 () Location of <i>removeDevice</i> operation (line number)	Step3 () Special instructions (if any) for assembling / disassembling this aggregate device (e.g. child order)	Step4 () FAILURE_ALARM logged

## Test Summary – OE\_TC\_029

Once testing is complete for every component of the OE under test, report the test result as follows:

Pass:	No failures detected
Fail:	Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.
Untested:	Condition which is not testable
N/A:	Not Applicable

**Overall Test Result** (Pass, Fail, Untested, or N/A): \_\_\_\_\_

OE0464 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.8. OE\_TC\_030 - AggregateDevice :: removeDevice raises InvalidObjectReference

**Test Case Number:** OE\_TC\_030

Device::removeDevice

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0465	The <i>removeDevice</i> operation shall raise the CF InvalidObjectReference when the input associatedDevice parameter is a nil CORBA object reference or does not exist in the AggregateDevice devices attribute.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pg. 3-75, Section 3.1.3.3.4.5.2.5

#### Test Objective

This test case verifies requirement OE0465. The objective of this test is to verify that the *removeDevice* operation raises the CF InvalidObjectReference when (1) the input associatedDevice parameter is a nil CORBA object reference or (2) the associatedDevice parameter does not exist in the AggregateDevice devices attribute.

#### Places to Verify

Aggregated Devices

#### IDL References

##### Exceptions

```
exception InvalidObjectReference { string msg; };
```

##### Operations

```
void removeDevice ( in CF::Device associatedDevice )  
    raises (CF::InvalidObjectReference);
```

## Preconditions

- The source code files are available.

## Test Description

A. Identify all aggregate devices. (OE0465)

1. **N/A:** No aggregate devices. If there are no aggregate devices, the test ends at this point.

For each aggregate device perform the following steps:

B. Verify the *removeDevice* method is present. (OE0465)

1. **Pass:** The *removeDevice* method is present.
2. **Fail:** The *removeDevice* method is not present.

C. Verify the *removeDevice* method throws an *InvalidObjectReference* exception if the associated Device is nil. (OE0465)

1. **Pass:** *InvalidObjectReference* exception is thrown
2. **Fail:** *InvalidObjectReference* exception is not thrown

D. Verify the *removeDevice* method throws an *InvalidObjectReference* exception, with the correct message format, if the associated Device is not currently in the Device sequence. (OE0465)

1. **Pass:** *InvalidObjectReference* exception is thrown
2. **Fail:** *InvalidObjectReference* exception is not thrown

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_030				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify all aggregate devices (OE0465).</b>				
1. Determine the association of parent and children for aggregated Device. Review the Software Assembly Descriptor (SAD) and source code. Record the source code file names and line numbers. If there are no aggregate devices, the test ends at this point.	<p>The system under test may or may not have aggregate devices.</p> <p>The requirement is not applicable (N/A) if there are no aggregate device implementations. (OE0465)</p>		In an aggregate (system) of devices there may be prescribed restrictions or orders that restrict how to perform the association and disassociation of child devices.	
<b>B. Verify that the <i>removeDevice</i> method is present (OE0465).</b>				
2. Verify that the <i>removeDevice</i> method is present.	<p><b>Pass:</b> The <i>removeDevice</i> method is present. (OE0465)</p> <p><b>Fail:</b> The <i>removeDevice</i> method is not present. (OE0465)</p>			
<b>C. Verify the <i>removeDevice</i> method throws an <i>InvalidObjectReference</i> exception if the associated Device is nil (OE0465).</b>				
3. Review the <i>removeDevice</i> code in the location where the associated Device is checked for a nil value.	<b>Fail:</b> The <i>removeDevice</i> operation does not check for a nil value. (OE0465)			
4. Verify the <i>removeDevice</i> method throws an <i>InvalidObjectReference</i> exception if the associated Device is nil.	<p><b>Pass:</b> <i>InvalidObjectReference</i> exception is thrown. (OE0465)</p> <p><b>Fail:</b> <i>InvalidObjectReference</i> exception is not thrown. (OE0465)</p>			
<b>D. Verify the <i>removeDevice</i> method throws an <i>InvalidObjectReference</i> exception if the associated Device is not currently in the Device sequence (OE0465).</b>				

OE_TC_030				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Review the <i>removeDevice</i> code in the location where the associated Device is compared to the known Device sequence.	<b>Fail:</b> The <i>removeDevice</i> operation does not compare the Device to the known Device sequence. (OE0465)			
6. Verify the <i>removeDevice</i> method throws an <i>InvalidObjectReference</i> exception if the associated Device not currently in the Device sequence.	<b>Pass:</b> <i>InvalidObjectReference</i> exception is thrown. (OE0465) <b>Fail:</b> <i>InvalidObjectReference</i> exception is not thrown. (OE0465)			
<b>End of Test</b>				

Test Recording Log – OE_TC_030						
Step 1 File and line number	Step 1 Parent Device	Step 1 Associated Device(s)	Step 2 & 3 Check for nil reference within <i>removeDevice</i> method	Step 4 Invalid Object Reference exception thrown Y/N	Step 5 Device compared to known device sequence Y/N	Step 6 Invalid Object Reference exception thrown Y/N



## Test Summary - OE\_TC\_030

Once testing is complete for every component of the OE under test, report the test result as follows:

Pass: No failures detected  
Fail: Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
Untested: Condition which is not testable  
N/A: Not Applicable

**Overall Test Result** (Pass, Fail, Untested, or N/A): \_\_\_\_\_

**OE0465** \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.9. OE\_TC\_058 - Device operationalState

#### Test Case Number: OE\_TC\_058

Device operationalState

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0400	The device shall send a StateChangeEvent Type event to the Incoming Domain Management event channel, whenever the operationalState attribute changes.
OE0400-C092	The <i>producerId</i> field is the identifier attribute of the device.
OE0400-C093	The <i>sourceId</i> field is the identifier attribute of the device.
OE0400-C094	The <i>stateChangeCategory</i> field is "OPERATIONAL STATE EVENT".
OE0400-C095	The <i>stateChangeFrom</i> field is the value of the operationalState attribute before the state change.
OE0400-C096	The <i>stateChangeTo</i> field is the value of the operationalState attribute after the state change.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-61, Section 3.1.3.3.1.4.3

#### Test Objective

This test case verifies OE0400 and the supporting criteria: OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, and OE0400-C096. The objective of this test case is to verify that upon the occurrence of a change of operationalState of the device, a StateChangeEvent Type event is sent to the Incoming Domain Management event channel. The StateChangeEvent Type event must contain current values for *producerId*, *sourceId*, *stateChangeCategory*, *stateChangeFrom* and *stateChangeTo* field.

#### Places to Verify

Devices

#### IDL References

##### Data

```
OperationalType { ENABLED, DISABLED };  
struct StateChangeEvent { string producerId; string sourceId;  
StandardEvent::StateChangeCategoryType stateChangeCategory;
```

StandardEvent::StateChangeType stateChangeFrom; StandardEvent::StateChangeType stateChangeTo; };

## Preconditions

- The Domain Profile files, including the DCD files, are available.
- The Domain Profile requirements have passed.
- The source code files of the Core Framework are available.

## Test Description

A. Determine the names of the devices of the Operating Environment and their associated Software Package Descriptor files (SPD) and source code files. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)

1. **Pass:** If devices exist for the Operating Environment, they can be determined as well as their associated SPD files and source code files located.
2. **N/A:** The devices of the operating environment are not available.

For all devices found in the Operating Environment:

B. Determine that the operational state can change. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)

1. **Pass:** The operational state changes during any of the device processes.
2. **N/A:** The operational state does not change during any of the device processes.

**NOTE:** If the design does not support an operational state change (e.g. always enabled), the requirement does not apply and the test ends. To get an N/A result, the device source code and documentation should be thoroughly inspected to see that the operational state never changes.

C. Verify that the device sends a StateChangeEvent event to the Incoming Domain Management Event Channel when the operationalState changes. (OE0400)

1. Verify that the device sends a StateChangeEvent event to the Incoming Domain Management Event Channel when the operationalState changes from DISABLED to ENABLED.
  - a. **Fail:** The device does not send an event to the Incoming Domain Management Event channel when the operationalState changes from DISABLED to ENABLED.
  - b. **Pass:** The device sends an event to the Incoming Domain Management Event channel when the operationalState changes from DISABLED to ENABLED.
  - c. **N/A:** A state change from DISABLED to ENABLED does not exist.
2. Verify that the device sends a StateChangeEvent event to the Incoming Domain Management Event Channel when the operationalState changes from ENABLED to DISABLED.

- a. **Fail:** The device does not send an event to the Incoming Domain Management Event channel when the operationalState changes from ENABLED to DISABLED.
  - b. **Pass:** The device sends an event to the Incoming Domain Management Event channel when the operationalState changes from ENABLED to DISABLED.
  - c. **N/A:** A state change from ENABLED to DISABLED does not exist.
- D. For all state changes, verify that the StateChangeEvent event sent by the device when the operationalState changes, contains current values for the *producerId*, *sourceId*, *StateChangeCategory*, *stateChangeFrom* field and *stateChangeTo* field. (OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)
- 1. Verify that the *producerId* of the StateChangeEvent stores the identifier attribute of the device. (OE0400-C092)
    - a. **Fail:** The *producerId* of the StateChangeEvent event does not contain the identifier attribute of the device.
    - b. **Pass:** The *producerId* of the StateChangeEvent event contains the identifier attribute of the device.
  - 2. Verify that the *sourceId* of the StateChangeEvent stores the identifier attribute of the device. (OE0400-C093)
    - a. **Fail:** The *sourceId* of the StateChangeEvent event does not contain the identifier attribute of the device.
    - b. **Pass:** The *sourceId* of the StateChangeEvent event contains the identifier attribute of the device.
  - 3. Verify that the *StateChangeCategory* field contains the value of OPERATIONAL\_STATE\_EVENT. (OE0400-C094)
    - a. **Fail:** The *StateChangeCategory* field does not contain the value, OPERATIONAL\_STATE\_EVENT.
    - b. **Pass:** The *StateChangeCategory* field contains the value, OPERATIONAL\_STATE\_EVENT.
  - 4. Verify that the *stateChangeFrom* field stores the value of the operationalState attribute before the state change. (OE0400-C095)
    - a. **Fail:** The *stateChangeFrom* field does not store the value of the operationalState attribute before the state change.
    - b. **Pass:** The *stateChangeFrom* field stores the value of the operationalState attribute before the state change.
  - 5. Verify that the *stateChangeTo* field stores the value of the operationalState attribute after the state change. (OE0400-C096)
    - a. **Fail:** The *stateChangeTo* field does not store the value of the operationalState attribute after the state change.
    - b. **Pass:** The *stateChangeTo* field stores the value of the operationalState attribute after the state change.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_058				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine the names of the devices of the Operating Environment and their associated Software Package Descriptor files (SPD) and source code files. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)</b>				
1. Examine the DCD file for each device name and record the name and SPD file of each device.	<p><b>Pass:</b> If devices exist for the Operating Environment, they can be determined as well as their associated SPD files and source code files located. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)</p> <p><b>N/A:</b> The devices of the OE are not available. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)</p>		<p>A device's id and name are usually specified in the DCD file as in the following example:</p> <pre>&lt;componentplacement&gt;   &lt;componentfileref     refid="CDATA"&gt;     &lt;componentinstantiation       id="DCE.ddd123xxxx"&gt;  &lt;usagename&gt;DeviceName&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt; &lt;componentfiles&gt; &lt;componentfile id="&lt;CDATA&gt; " type="SPD"&gt;   &lt;localfile     name=".../logservice/DeviceFile.spd.xml"/&gt;   &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre>	

OE_TC_058				
Steps	Expected Results	Actual Results	Comments	Test Result
2. Locate the source code for the device and record the name of the file.	<p><b>Pass:</b> If devices exist for the Operating Environment, they can be determined as well as their associated SPD files and source code files located.(OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)</p> <p><b>Untested:</b> The source code of the devices is not located. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)</p>		The development engineer and the developer documentation, if available, is the best source for the location of the source code.	
<p><b>B. Determine that the operational state can change. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)</b></p> <p><b>NOTE: If the design does not support an operational state change(e.g. always enabled), the requirement does not apply and the test ends. To get an N/A result, the device source code and documentation should be thoroughly inspected to see that the operational state never changes.</b></p>				
3. Verify in the source code that the device contains the operationState attribute.	<p><b>Pass:</b> The operationalState attribute for the device exists.(OE0400)</p> <p><b>Fail:</b> The operationalState attribute for the device does not exist. (OE0400)</p>			
4. Verify in the source code that the operational state can change.	<p><b>N/A:</b> The operational state does not change during any of the device processes. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)</p>		Note: this may involve inspection of documentation to find situations where a device changes operationalState.	

OE_TC_058				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>C. Verify that the device sends a StateChangeEventType event to the Incoming Domain Management Event Channel when the operationalState changes. (OE0400)</b> <b>1. Verify that the device sends a StateChangeEventType event to the Incoming Domain Management Event Channel when the operationalState changes from ENABLED to DISABLED. (OE0400)</b> <b>2. Verify that the device sends a StateChangeEventType event to the Incoming Domain Management Event Channel when the operationalState changes from DISABLED to ENABLED. (OE0400)</b>				
5. Verify in the source code that the device contains logic to trigger the sending of an event to the Incoming Domain Management Event channel.	<b>Pass:</b> Device contains source code that sends an event to the incoming event channel. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)  <b>Fail:</b> Device does not contain source code that sends an event to the incoming event channel. (OE0400, OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)		Failure of any of these criteria OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095 or OE0400-C096 means failure of the requirement OE0400.	

OE_TC_058				
Steps	Expected Results	Actual Results	Comments	Test Result
6. Verify in the source code that the device sends an event to the incoming Domain Management Event Channel when the operationalState changes from DISABLED to ENABLED.	<p><b>Pass:</b> The device sends an event to the Incoming Domain Management Event channel when the operationalState changes from DISABLED to ENABLED. (OE0400)</p> <p><b>Fail:</b> The device does not send an event to the Incoming Domain Management Event channel when the operationalState changes to from DISABLED to ENABLED. (OE0400)</p> <p><b>N/A:</b> A state change from DISABLED to ENABLED does not exist. (OE0400)</p>		Possibly need to look at the setOperationalState(...) code of the device.	



OE_TC_058				
Steps	Expected Results	Actual Results	Comments	Test Result
7. Verify in the source code that the device sends an event to the incoming Domain Management Event Channel when the operationalState changes from ENABLED to DISABLED.	<p><b>Pass:</b> The device sends an event to the Incoming Domain Management Event channel when the operationalState changes from ENABLED to DISABLED. (OE0400)</p> <p><b>Fail:</b> The device does not send an event to the Incoming Domain Management Event channel when the operationalState changes from ENABLED to DISABLED. (OE0400)</p> <p><b>N/A:</b> A state change from ENABLED to DISABLED does not exist. (OE0400)</p>		Possibly need to look at the setOperationalState(...) code of the device.	
<p><b>D. For all state changes, verify that the StateChangeEvent event sent by the device when the operationalState changes contains current values for the producerId, sourceId, StateChangeCategory, stateChangeFrom field and stateChangeTo field. (OE0400-C092, OE0400-C093, OE0400-C094, OE0400-C095, OE0400-C096)</b></p> <p>1. Verify that the producerId of the StateChangeEvent stores the identifier attribute of the device. (OE0400-C092)</p> <p>2. Verify that the sourceId of the StateChangeEvent stores the identifier attribute of the device. (OE0400-C093)</p> <p>3. Verify that the StateChangeCategory field contains the value of OPERATIONAL_STATE_EVENT. (OE0400-C094)</p> <p>4. Verify that the stateChangeFrom field stores the value of the operationalState attribute before the state change. (OE0400-C095)</p> <p>5. Verify that the stateChangeTo field stores the value of the operationalState attribute after the state change. (OE0400-C096)</p>				
8. Verify in the source code that producerId of the StateChangeEvent stores the identifier attribute of the device.	<p><b>Fail:</b> The <i>producerId</i> is not set to the identifier attribute of the device. (OE0400-C092)</p> <p><b>Pass:</b> The <i>producerId</i> is set to the identifier attribute of the device. (OE0400-C092)</p>		Failure of this criterion OE0400-C092 means failure of the requirement OE0400.	

OE_TC_058				
Steps	Expected Results	Actual Results	Comments	Test Result
9. Verify in the source code that the <i>sourceId</i> of the <i>StateChangeEvent</i> stores the identifier attribute of the device.	<b>Fail:</b> The <i>sourceId</i> is not set to the identifier attribute of the device. (OE0400-C093)  <b>Pass:</b> The <i>sourceId</i> is set to the identifier attribute of the device. (OE0400-C093)		Failure of this criterion OE0400-C093 means failure of the requirement OE0400.	
10. Verify in the source code that the <i>StateChangeCategory</i> field contains the value of <i>OPERATIONAL_STATE_EVENT</i> .	<b>Fail:</b> The <i>StateChangeCategory</i> field does not contain the value of <i>OPERATIONAL_STATE_EVENT</i> . (OE0400-C094)  <b>Pass:</b> The <i>StateChangeCategory</i> field contains the value of <i>OPERATIONAL_STATE_EVENT</i> . (OE0400-C094)		Failure of this criterion OE0400-C094 means failure of the requirement OE0400.	
11. Verify that the <i>stateChangeFrom</i> field contains the value of the <i>operationalState</i> attribute before the state change.	<b>Fail:</b> The <i>stateChangeFrom</i> field does not contain the value of the <i>operationalState</i> attribute before the state change. (OE0400-C095)  <b>Pass:</b> The <i>stateChangeFrom</i> field contains the value of the <i>operationalState</i> attribute before the state change. (OE0400-C095)		Failure of this criterion OE0400-C095 means failure of the requirement OE0400.	

OE_TC_058				
Steps	Expected Results	Actual Results	Comments	Test Result
12. Verify that the <i>stateChangeTo</i> field contains the value of the <i>operationalState</i> attribute after the state change.	<b>Fail:</b> The <i>stateChangeTo</i> field does not contain the value of the <i>operationalState</i> attribute after the state change. (OE0400-C096)  <b>Pass:</b> The <i>stateChangeTo</i> field contains the value of the <i>operationalState</i> attribute after the state change. (OE0400-C096)		Failure of this criterion OE0400-C096 means failure of the requirement OE0400.	
<b>End of Test</b>				

Test Recording Log – OE_TC_058											
Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step10	Step11	Step12
List of names of devices.	List of source files of devices.	Device(s) contains operational State attribute.	Device operational State can change.	Device contains logic to produce StateChangeEvent.	Event sent for state change from DISABLED to ENABLED.	Event is sent for state change from ENABLED to DISABLED.	ProducerId of StateChangeEvent Type contains current value.	SourceId of StateChangeEvent Type contains OPERATIONAL_STATE_EVENT.	StateChangeEvent Category field of StateChangeEvent Type contains current value.	StateChangeEventFrom field contains value of operational State before change.	StateChangeEventTo field contains value of operational State after change.

## Test Summary OE\_TC\_058

Once testing is complete for every component of the OE under test, report the test result as follows:

Pass: No failures detected  
Fail: Failure(s) detected in Step(s)(x) Failure of any OE0400 criteria results in a failure of OE0400.  
Untested: Condition which is not testable  
N/A: Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0400 \_\_\_\_\_

OE0400-C092 \_\_\_\_\_

OE0400-C093 \_\_\_\_\_

OE0400-C094 \_\_\_\_\_

OE0400-C095 \_\_\_\_\_

OE0400-C096 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_

Test Engineer: \_\_\_\_\_

Date Tested: \_\_\_\_\_

Witness: \_\_\_\_\_

### B.3.10. OE\_TC\_062 - Base Device Interfaces

#### Test Case Number: OE\_TC\_062

Base Device Interfaces

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0701	Base Device Interfaces shall be implemented using the CF IDL presented in Appendix C.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-57 through 3-58, Section 3.1.3.3 and 3.1.3.3.1.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	All
SCA Appendix C: Core Framework IDL Attachment 1	Version 2.2.2 15 May 2006	All
C++ Language Mapping	Version 1.2, OMG Document Number: formal/2008-01-10	Page 5, Section 4.1

#### Test Objective

This test case verifies OE0701. The objective of this test is to verify that Base Device Interfaces are implemented using the CF IDL as presented in Appendix C. The CF IDL implements the CF (Core Framework) module. The Base Device Interfaces and the interfaces they inherit in the CF module are:

Interface Table	
Interface	Interfaces Inherited
Device	Resource
LoadableDevice	Device
ExecutableDevice	LoadableDevice
Resource	LifeCycle, TestableObject, PropertySet, PortSupplier

#### Places to Verify

Developer IDL and source code

## IDL References

None.

## Preconditions

- The Device source code files are available.
- The developer IDL files are available.

## Test Description

Note: This test should be run concurrently with OE\_TC\_116.

- A. Locate the CF IDL used by the developer and verify that it matches the CF IDL from attachment 1 to Appendix C of the SCA. (OE0701)
  1. **Pass:** The CF IDL supplied by the developer matches the CF IDL in attachment 1.
  2. **Fail:** No CF IDL is supplied.
  3. **Fail:** The CF IDL supplied by the developer does not match the CF IDL in attachment 1.
- B. Verify that the skeleton code for Base Device Interfaces was generated from the CF IDL. (OE0701)
  1. **Pass:** The skeleton code was generated from the CF IDL.
  2. **Fail:** The skeleton code was not generated from the CF IDL.
- C. Verify that the definition of the Device interface in product's CF.IDL and SCA document are identical. (OE0701)
  1. **Pass:** The product's CF.IDL is identical to the SCA document version of the IDL.
  2. **Fail:** The product's CF.IDL is not identical to the SCA document version of the IDL.
- D. Verify that the class CF::Device inherits from the class POA\_CF::Device, which is in CF\_s.h. (OE0701)
  1. **Pass:** The class CF::Device inherits from the class POA\_CF::Device, which is in CF\_s.h.
  2. **Fail:** The class CF::Device does not inherit from the class POA\_CF::Device, which is in CF\_s.h.
- E. Verify that the signatures of all SCA-defined operations of the Device interface are compliant with the IDL file. (OE0701)
  1. **Pass:** The signatures of all SCA-defined operations of the Device interface are compliant with the IDL file.
  2. **Fail:** The signatures of all SCA-defined operations of the Device interface are not compliant with the IDL file.

## Manual Test Steps

Notes:

1. Test Result will include Pass, Fail, Untested, or N/A.
2. The Test Recording Log is intended to record data for each step that requires recording of data.
3. This test should be run concurrently with OE\_TC\_116.

OE_TC_062				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate the CF IDL used by the developer and verify that it matches the CF IDL from attachment 1 to Appendix C of the SCA. (OE0701)</b>				
1. Locate the CF IDL supplied by the developer and record its location.	<b>Pass:</b> The CF IDL is located. (OE0701)  <b>Fail:</b> The CF IDL is not located. (OE0701)			
2. Using a diff tool, compare the CF IDL supplied with the SCA CF IDL from attachment 1 of appendix C of the SCA.	<b>Pass:</b> The developer supplied CF IDL and the SCA CF IDL match. (OE0701)  <b>Fail:</b> The CF IDL does not match. (OE0701)		The attachment 1 is the compilable form of the CF IDL. White space and comment differences do not constitute a failure. Rearrangement of the CF IDL also may not constitute a failure. Any differences found need to be analyzed to determine if they may cause a functional difference in the resulting code.	
<b>B. Verify that the skeleton code for Base Device Interfaces was generated from the CF IDL. (OE0701)</b>				
3. Search the supplied source code header files for "namespace CF" and verify that it exists.	<b>Pass:</b> "namespace CF" is found. (OE0701)  <b>Fail:</b> "namespace CF" is not found. (OE0701)		For C++ a module in an IDL translates to a namespace. For Java or Ada, it would be a package. The search string will have to be modified appropriately.	
<b>C. Verify that the definition of the Device interface in product's CF.IDL and SCA document are identical. (OE0701)</b>				



OE_TC_062				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Compare the Device interface in product's CF.IDL with that of the SCA documented IDL.	<b>Pass:</b> The two IDLs are identical. (OE0701)  <b>Fail:</b> "namespace CF" is not found. (OE0701)			
<b>D. Verify that the class CF::Device inherits from the class POA_CF::Device, which is in CF_s.h. (OE0701)</b>				
5. Verify the CF Device class inherits from the POA_CF::Device class, found in CF_s.h	<b>Pass:</b> The CF::Device class inherits from the POA_CF::Device class. (OE0701)  <b>Fail:</b> The CF::Device class does not inherit from the POA_CF::Device class. (OE0701)			
<b>E. Verify that the signatures of all SCA-defined operations of the Device interface are compliant with the IDL file. (OE0701)</b>				
6. Verify that the signatures of all SCA defined operations of the Device interface are compliant with the IDL file.	<b>Pass:</b> The signatures of all SCA defined operations of the Device interface are compliant with the IDL file. (OE0701)  <b>Fail:</b> The signatures of all SCA defined operations of the Device interface are not compliant with the IDL file. (OE0701)			
<b>End of Test</b>				

Test Recording Log OE_TC_062		
Step 1 (CF idl location)	Step 2 (matches)	Step 3 (idl module maps properly – header file name)

## Test Summary OE\_TC\_062

Once testing is complete for every component of OE under test, report the test result as follows:

Pass: No failures detected  
Fail: Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
Untested: Condition which is not testable  
N/A: Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0701\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Test Engineer: \_\_\_\_\_

Date Tested: \_\_\_\_\_

Witness: \_\_\_\_\_

### B.3.11. OE\_TC\_079 - Device :: usageState

#### Test Case Number: OE\_TC\_079

Device::usageState and other attributes

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0345	The readonly <i>usageState</i> attribute shall contain the device's usage state (IDLE, ACTIVE, or BUSY).
OE0381	The device shall send a <i>StateChangeEvent</i> event to the Incoming Domain Management event channel, whenever the <i>usageState</i> attribute changes.
OE0381-C082	The <i>producerId</i> field is the identifier attribute of the device.
OE0381-C083	The <i>sourceId</i> field is the identifier attribute of the device.
OE0381-C084	The <i>stateChangeCategory</i> field is "USAGE_STATE_EVENT".
OE0381-C085	The <i>stateChangeFrom</i> field is the value of the <i>usageState</i> attribute before the state change.
OE0381-C086	The <i>stateChangeTo</i> field is the value of the <i>usageState</i> attribute after the state change.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-60, Section 3.1.3.3.1.4.1; Page 3-63 Section 3.1.3.3.1.5.2.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-25 Page C-27 Page C-37

#### Test Objective

This test case verifies OE0345, OE0381 and its supporting criteria. The objective of this test case is to verify the device's *usageState* attribute. Furthermore, this test case is to verify that a device sends a *StateChangeEvent* event to the Incoming Domain Management event channel (IDM\_CHANNEL) when the *usageState* attribute is changed by the *allocateCapacity* and *deallocateCapacity* operations. The test case must also verify the criteria associated with this requirement. The criteria ensure that the *StateChangeEvent* event is populated with the *producerId*, *sourceId*, *stateChangeCategory* as USAGE\_STATE\_EVENT, *stateChangeFrom* field and *stateChangeTo* field.

#### Places to Verify

Devices

## IDL References

### Data

```
enum UsageType { IDLE, ACTIVE, BUSY };
readonly attribute CF::Device::UsageType usageState;
struct StateChangeEvent { string producerId;
                        string sourceId;
                        StandardEvent::StateChangeCategoryType stateChangeCategory;
                        StandardEvent::StateChangeType stateChangeFrom;
                        StandardEvent::StateChangeType stateChangeTo; }
```

### Exceptions

```
exception InvalidCapacity { string msg; CF::Properties capacities; };
exception InvalidState { string msg; };
```

### Operations

```
Boolean allocateCapacity ( in CF::Properties capacities ) raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);
void deallocateCapacity ( in CF::Properties capacities ) raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);
```

## Preconditions

- The Domain Profile Files are available
- The source code files of the Core Framework(CF) are available

## Test Description

A. Identify the devices of the Core Framework. (OE0381)

1. **Pass:** Devices are present in the Core Framework.
2. **Fail:** Devices are not present in the Core Framework.
3. **Untested:** The source code of the devices is not available.

For each device implementation found:

B. Verify that the readonly attribute named “usageState” contains the device’s usage state. (OE0345)

1. **Pass:** The readonly attribute named “usageState” contains the device’s usage state.
2. **Fail:** The readonly attribute named “usageState” is not of type UsageType.
3. **Fail:** The readonly attribute named “usageState” does not contain the device’s usage state.

- C. Verify that the *allocateCapacity* and *deallocateCapacity* operations exist for the device. (OE0381)
1. **Fail:** The *allocateCapacity* operation does not exist for the device.
  2. **Fail:** The *deallocateCapacity* operation does not exist for the device.
- D. Verify that the device sends a *StateChangeEvent* event to the *IDM\_CHANNEL* when the *allocateCapacity* operation changes the *usageState* attribute. (OE0381)
- Note:** The possible state changes from the *allocateCapacity* operation are ACTIVE to BUSY, IDLE to ACTIVE, IDLE to BUSY.
1. **Pass:** The device sends a *StateChangeEvent* event to the *IDM\_CHANNEL* when the *allocateCapacity* operation changes the *usageState* attribute.
  2. **Fail:** The device does not send a *StateChangeEvent* event to the *IDM\_CHANNEL* when the *allocateCapacity* operation changes the *usageState* attribute.
- E. For all state changes, verify that the *StateChangeEvent* event sent by the device contains current values for the *producerId*, *sourceId*, *StateChangeCategory*, *stateChangeFrom* field and *stateChangeTo* field when the *allocateCapacity* operation changes the *usageState*.
1. Verify that the *producerId* of the *StateChangeEvent* stores the identifier attribute of the device (OE0381-C082).
    - a. **Pass:** The *producerId* of the *StateChangeEvent* event contains the identifier attribute of the device.
    - b. **Fail:** The *producerId* of the *StateChangeEvent* event does not contain the identifier attribute of the device.
  2. Verify that the *sourceId* of the *StateChangeEvent* stores the identifier attribute of the device (OE0381-C083).
    - a. **Pass:** The *sourceId* of the *StateChangeEvent* event contains the identifier attribute of the device.
    - b. **Fail:** The *sourceId* of the *StateChangeEvent* event does not contain the identifier attribute of the device.
  3. Verify that the *StateChangeCategory* field contains the value of *USAGE\_STATE\_EVENT* (OE0381-C084).
    - a. **Pass:** The *StateChangeCategory* field contains the value, *USAGE\_STATE\_EVENT*.
    - b. **Fail:** The *StateChangeCategory* field does not contain the value, *USAGE\_STATE\_EVENT*.
  4. Verify that the *stateChangeFrom* field stores the value of the *usageState* attribute before the state change (OE0381-C085).
    - a. **Pass:** The *stateChangeFrom* field stores the value of the *usageState* attribute before the state change.
    - b. **Fail:** The *stateChangeFrom* field does not store the value of the *usageState* attribute before the state change.
  5. Verify that the *stateChangeTo* field stores the value of the *usageState* attribute after the state change (OE0381-C086).
    - a. **Pass:** The *stateChangeTo* field stores the value of the *usageState* attribute after the state change.
    - b. **Fail:** The *stateChangeTo* field does not store the value of the *usageState* attribute after the state change.
- F. Verify that the device sends a *StateChangeEvent* event to the *IDM\_CHANNEL* when the *deallocateCapacity* operation changes the *usageState*. (OE0381)
- Note:** The possible state changes for the *deallocateCapacity* operation are: BUSY to IDLE, BUSY to ACTIVE, ACTIVE to IDLE.
1. **Pass:** The device sends a *StateChangeEvent* event to the *IDM\_CHANNEL* when the *deallocateCapacity* operation changes the *usageState* attribute.

- 
2. **Fail:** The device does not send a `StateChangeEvent` event to the `IDM_CHANNEL` when the *deallocateCapacity* operation changes the *usageState* attribute.
- G. For all state changes, verify that the `StateChangeEvent` event sent by the device contains current values for the *producerId*, *sourceId*, *StateChangeCategory*, *stateChangeFrom* field and *stateChangeTo* field when the *deallocateCapacity* operation changes the *usageState* (OE0381, OE0381-C082, OE0381-C083, OE0381-C084, OE0381-C085, OE0381-C086)
1. Verify that the *producerId* of the `StateChangeEvent` stores the identifier attribute of the device.
    - a. **Pass:** The *producerId* of the `StateChangeEvent` event contains the identifier attribute of the device.
    - b. **Fail:** The *producerId* of the `StateChangeEvent` event does not contain the identifier attribute of the device.
  2. Verify that the *sourceId* of the `StateChangeEvent` stores the identifier attribute of the device.
    - a. **Pass:** The *sourceId* of the `StateChangeEvent` event contains the identifier attribute of the device.
    - b. **Fail:** The *sourceId* of the `StateChangeEvent` event does not contain the identifier attribute of the device.
  3. Verify that the *StateChangeCategory* field contains the value of `USAGE_STATE_EVENT`.
    - a. **Pass:** The *StateChangeCategory* field contains the value, `USAGE_STATE_EVENT`.
    - b. **Fail:** The *StateChangeCategory* field does not contain the value, `USAGE_STATE_EVENT`.
  4. Verify that the *stateChangeFrom* field stores the value of the *usageState* attribute before the state change.
    - a. **Pass:** The *stateChangeFrom* field stores the value of the *usageState* attribute before the state change.
    - b. **Fail:** The *stateChangeFrom* field does not store the value of the *usageState* attribute before the state change.
  5. Verify that the *stateChangeTo* field stores the value of the *usageState* attribute after the state change.
    - a. **Pass:** The *stateChangeTo* field stores the value of the *usageState* attribute after the state change.
    - b. **Fail:** The *stateChangeTo* field does not store the value of the *usageState* attribute after the state change.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_079				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the devices of the Core Framework. (OE0345, OE0381)</b>				
1. Identify the devices of the Core Framework either through developer conversions, CF documentation, or through examination of the Domain Profile XML files.	<p><b>Pass:</b> Devices are present in the CF. (OE0345, OE0381)</p> <p><b>Fail:</b> Devices are not present in the CF. (OE0345, OE0381)</p> <p><b>Untested:</b> Devices are provided for the CF but there source code files are not available. (OE0345, OE0381)</p>		<p>May need to consult with the developer for the location of the devices.</p> <p>A device's id and name are usually specified in the DCD file as in the following example:</p> <pre>&lt;componentplacement&gt;   &lt;componentfileref refid="xxxx"/&gt;   &lt;componentinstantiation     id="DCE:xxxx-yyyyy-zzzz"&gt;     &lt;usagename&gt;zzzz&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt; &lt;componentfiles&gt; &lt;componentfile id="xxxx " type="SPD"&gt;   &lt;localfile name=".../yyyyy.spd.xml"/&gt; &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre> <p>Note: The usagename is usually the name of the device.</p> <p>The SCD file of a device component will have a componenttype element value-equaling device.</p>	
2. Locate the source code for the logical device and record the name of the file.	<b>Untested:</b> The source code for the devices is not located. (OE0345, OE0381)		Using the clue of the componentplacementref element (Step 2) and the information given inside the SPD file could also give additional information on how to locate the source code.	
<b>For each device implementation found in step A, perform the following:</b>				



OE_TC_079				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>B. Verify that the readonly attribute named “usageState” contains the device’s usage state. (OE0345)</b>				
3. Verify that there is a readonly attribute named “usageState” that is of type UsageType.	<b>Pass:</b> A read-only UsageType attribute named “usageState” exists. (OE0345)  <b>Fail:</b> A read-only UsageType attribute named “usageState” does not exist. (OE0345)		The definition of UsageType is part of the IDL and should be defined as an enum with the values IDLE, ACTIVE and BUSY only.	
4. Verify that the attribute named “usageState” contains the device’s usage state.	<b>Pass:</b> A UsageType attribute named “usageState” contains the device’s usage state. (OE0345)  <b>Fail:</b> A UsageType attribute named “usageState” does not contain the device’s usage state. (OE0345)			
<b>C. Verify that the <i>allocateCapacity</i> and <i>deallocateCapacity</i> operations exist for the device. (OE0381)</b>				
5. Verify in the source code of the device that the <i>allocateCapacity</i> and the <i>deallocateCapacity</i> operations are implemented.	<b>Pass:</b> The <i>allocateCapacity</i> and <i>deallocateCapacity</i> operations exist for the device. (OE0381)  <b>Fail:</b> The <i>allocateCapacity</i> or the <i>deallocateCapacity</i> operations do not exist for the device. (OE0381)		Developer may have one or more device classes with the <i>allocateCapacity</i> operation. The devices may inherit this functionality from parent classes. This is acceptable.	
<b>D. Verify that the device sends a StateChangeEventType event to the IDM_CHANNEL when the <i>allocateCapacity</i> operation changes the <i>usageState</i>. (OE0381)</b> <b>Note:</b> The possible state changes from the <i>allocateCapacity</i> operation are ACTIVE to BUSY, IDLE to ACTIVE, IDLE to BUSY.				

OE_TC_079				
Steps	Expected Results	Actual Results	Comments	Test Result
6. Verify in the source code of the device that a <code>StateChangeEvent</code> event is sent to the <code>IDM_CHANNEL</code> when the <code>allocateCapacity</code> operation changes the <code>usageState</code> from the previous state: which could be either, <code>ACTIVE</code> to <code>BUSY</code> , <code>IDLE</code> to <code>ACTIVE</code> or <code>IDLE</code> to <code>BUSY</code> .	<p><b>Pass:</b> The device sends an event to the <code>IDM_CHANNEL</code> when the <code>allocateCapacity</code> operation changes the value of the <code>usageState</code> attribute. (OE0381)</p> <p><b>Fail:</b> The device does not send an event to the <code>IDM_CHANNEL</code> when the <code>allocateCapacity</code> operation changes the <code>usageState</code> value. (OE0381)</p>		<p>Note the possible state change transitions are:</p> <ol style="list-style-type: none"> <li>1) <code>ACTIVE</code> to <code>BUSY</code></li> <li>2) <code>IDLE</code> to <code>ACTIVE</code></li> <li>3) <code>IDLE</code> to <code>BUSY</code></li> </ol>	
7. Verify in the source code of the <code>allocateCapacity</code> operation that the <code>StateChangeEvent</code> event mentioned in step 5 is sent to the <code>IDM_CHANNEL</code> using the <code>CosEventComm::PushConsumer</code> interface.	<p><b>Pass:</b> The device sends an event to the <code>IDM_CHANNEL</code> using the <code>CosEventComm::PushConsumer</code> interface. (OE0381)</p> <p><b>Fail:</b> The device does not send the event using the <code>CosEventComm::PushConsumer</code> interface. (OE0381)</p> <p><b>Fail:</b> The device does not send the event to the <code>IDM_CHANNEL</code>. (OE0381)</p>		<p>The operation call to look for in the source code will be similar to:</p> <pre>CosEventComm::PushConsumer::push()</pre>	
<p><b>E. For all state changes, verify that the <code>StateChangeEvent</code> event sent by the device contains current values for the <code>producerId</code>, <code>sourceId</code>, <code>StateChangeCategory</code>, <code>stateChangeFrom</code> field and <code>stateChangeTo</code> field when the <code>allocateCapacity</code> operation changes the <code>usageState</code>. OE0381, OE0381-C082, OE0381-C083, OE0381-C084, OE0381-C085, OE0381-C086)</b></p> <ol style="list-style-type: none"> <li>1. Verify that the <code>producerId</code> of the <code>StateChangeEvent</code> stores the identifier attribute of the device. (OE0381, OE0381-C082)</li> <li>2. Verify that the <code>sourceId</code> of the <code>StateChangeEvent</code> stores the identifier attribute of the device. (OE0381, OE0381-C083)</li> <li>2. Verify that the <code>StateChangeCategory</code> field contains the value of <code>USAGE_STATE_EVENT</code>. (OE0381, OE0381-C084)</li> <li>1. Verify that the <code>stateChangeFrom</code> field stores the value of the <code>usageState</code> attribute before the state change. (OE0381, OE0381-C085)</li> <li>2. Verify that the <code>stateChangeTo</code> field stores the value of the <code>usageState</code> attribute after the state change. (OE0381, OE0381-C086)</li> </ol>				

OE_TC_079				
Steps	Expected Results	Actual Results	Comments	Test Result
8. Verify in the source code that the <i>producerId</i> of the <i>StateChangeEvent</i> event is the identifier attribute of the device when the <i>allocateCapacity</i> operation changes the <i>usageState</i> .	<b>Pass:</b> The <i>producerId</i> of the <i>StateChangeEvent</i> event contains the identifier attribute of the device. (OE0381-C082)  <b>Fail:</b> The <i>producerId</i> of the <i>StateChangeEvent</i> event does not contain the identifier attribute of the device. (OE0381-C082)		Failure of this criterion OE0381-C082 means failure of the requirement OE0381	
9. Verify in the source code that the <i>sourceId</i> of the <i>StateChangeEvent</i> event is the identifier attribute of the device when the <i>allocateCapacity</i> operation changes the <i>usageState</i> .	<b>Pass:</b> The <i>sourceId</i> of the <i>StateChangeEvent</i> event contains the identifier attribute of the device. (OE0381-C083)  <b>Fail:</b> The <i>sourceId</i> of the <i>StateChangeEvent</i> event does not contain the identifier attribute of the device. (OE0381-C083)		Failure of this criterion OE0381-C083 means failure of the requirement OE0381	
10. Verify in the source code that the <i>StateChangeCategory</i> field of the device contains the value of <i>USAGE_STATE_EVENT</i> device when the <i>allocateCapacity</i> operation changes the <i>usageState</i> .	<b>Pass:</b> The <i>StateChangeCategory</i> field contains the value, <i>USAGE_STATE_EVENT</i> . (OE0381-C084)  <b>Fail:</b> The <i>StateChangeCategory</i> field does not contain the value, <i>USAGE_STATE_EVENT</i> . (OE0381-C084)		Failure of this criterion OE0381-C084 means failure of the requirement OE0381	
11. Verify in the source code that the <i>stateChangeFrom</i> field stores the value of the <i>usageState</i> attribute before the state change device when the <i>allocateCapacity</i> operation changes the <i>usageState</i> .	<b>Pass:</b> The <i>stateChangeFrom</i> field stores the value of the <i>usageState</i> attribute before the state change. (OE0381-C085)  <b>Fail:</b> The <i>stateChangeFrom</i> field does not store the value of the <i>usageState</i> attribute before the state change. (OE0381-C085)		Failure of this criterion OE0381-C085 means failure of the requirement OE0381	

OE_TC_079				
Steps	Expected Results	Actual Results	Comments	Test Result
12. Verify in the source code that the <i>stateChangeTo</i> field stores the value of the <i>usageState</i> attribute after the state change device when the <i>allocateCapacity</i> operation changes the <i>usageState</i> .	<p><b>Pass:</b> The <i>stateChangeTo</i> field stores the value of the <i>usageState</i> attribute after the state change. (OE0381-C086)</p> <p><b>Fail:</b> The <i>stateChangeTo</i> field does not store the value of the <i>usageState</i> attribute after the state change. (OE0381-C086)</p>		Failure of this criterion OE0381-C086 means failure of the requirement OE0381	
<p><b>F. Verify that the device sends a StateChangeEvent event to the IDM_CHANNEL when the deallocateCapacity operation changes the usageState. (OE0381)</b></p> <p><b>Note: The possible state changes for the deallocateCapacity operation are: BUSY to IDLE, BUSY to ACTIVE, ACTIVE to IDLE.</b></p>				
13. Verify in the source code that the device sends a StateChangeEvent event to the IDM_CHANNEL when the <i>deallocateCapacity</i> operation changes the <i>usageState</i> attribute value from the previous value, which could be either BUSY to IDLE, BUSY to ACTIVE, or ACTIVE to IDLE.	<p><b>Pass:</b> The device sends an event to the Incoming Domain Management Event channel when the <i>deallocateCapacity</i> operation changes the value of the <i>usageState</i> attribute. (OE0381)</p> <p><b>Fail:</b> The device does not send an event to the Incoming Domain Management Event channel when the <i>deallocateCapacity</i> operation changes the value of the <i>usageState</i> attribute. (OE0381)</p>		<p>Note the possible state change transitions are:</p> <ol style="list-style-type: none"> <li>1) BUSY to IDLE</li> <li>2) BUSY to ACTIVE</li> <li>3) ACTIVE to IDLE</li> </ol>	

OE_TC_079				
Steps	Expected Results	Actual Results	Comments	Test Result
14. Verify in the source code of the <i>deallocateCapacity</i> operation that the <i>StateChangeEvent</i> event mentioned in step 12 is sent to the <i>IDM_CHANNEL</i> using the <i>CosEventComm::PushConsumer</i> interface.	<p><b>Pass:</b> The device sends an event to the <i>IDM_CHANNEL</i> using the <i>CosEventComm::PushConsumer</i> interface. (OE0381)</p> <p><b>Fail:</b> The device does not send the event using the <i>CosEventComm::PushConsumer</i> interface. (OE0381)</p> <p><b>Fail:</b> The device does not send the event to the <i>IDM_CHANNEL</i>. (OE0381)</p>		The operation call to look for in the source code will be similar to: <i>CosEventComm::PushConsumer::push()</i>	
<p><b>G. For all state changes, verify that the <i>StateChangeEvent</i> event sent by the device contains current values for the <i>producerId</i>, <i>sourceId</i>, <i>StateChangeCategory</i>, <i>stateChangeFrom</i> field and <i>stateChangeTo</i> field when the <i>deallocateCapacity</i> operation changes the <i>usageState</i>. (OE0381-C082, OE0381-C083, OE0381-C084, OE0381-C085, OE0381-C086)</b></p> <ol style="list-style-type: none"> <li>1. Verify that the <i>producerId</i> of the <i>StateChangeEvent</i> contains the identifier attribute of the device. (OE0381-C082)</li> <li>2. Verify that the <i>sourceId</i> of the <i>StateChangeEvent</i> contains the identifier attribute of the device. (OE0381-C083)</li> <li>3. Verify that the <i>StateChangeCategory</i> field contains the value of <i>USAGE_STATE_EVENT</i>. (OE0381-C084)</li> <li>4. Verify that the <i>stateChangeFrom</i> field stores the value of the <i>usageState</i> attribute before the state change. (OE0381-C085)</li> <li>5. Verify that the <i>stateChangeTo</i> field stores the value of the <i>usageState</i> attribute after the state change. (OE0381-C086)</li> </ol>				
15. Verify in the source code that the <i>producerId</i> of the <i>StateChangeEvent</i> event is the identifier attribute of the device when the <i>deallocateCapacity</i> operation changes the <i>usageState</i> .	<p><b>Pass:</b> The <i>producerId</i> of the <i>StateChangeEvent</i> event contains the identifier attribute of the device. (OE0381-C082)</p> <p><b>Fail:</b> The <i>producerId</i> of the <i>StateChangeEvent</i> event does not contain the identifier attribute of the device. (OE0381-C082)</p>		Failure of this criterion OE0381-C082 means failure of the requirement OE0381	

OE_TC_079				
Steps	Expected Results	Actual Results	Comments	Test Result
16. Verify in the source code that the sourceId of the StateChangeEvent event is the identifier attribute of the device when the deallocateCapacity operation changes the usageState.	<p><b>Pass:</b> The <i>sourceId</i> of the StateChangeEvent event contains the identifier attribute of the device. (OE0381-C083)</p> <p><b>Fail:</b> The <i>sourceId</i> of the StateChangeEvent event does not contain the identifier attribute of the device. (OE0381-C083)</p>		Failure of this criterion OE0381-C083 means failure of the requirement OE0381	
17. Verify in the source code that the StateChangeCategory field of the device contains the value of USAGE_STATE_EVENT device when the deallocateCapacity operation changes the usageState.	<p><b>Pass:</b> The <i>StateChangeCategory</i> field contains the value, USAGE_STATE_EVENT. (OE0381-C084)</p> <p><b>Fail:</b> The <i>StateChangeCategory</i> field does not contain the value, USAGE_STATE_EVENT. (OE0381-C084)</p>		Failure of this criterion OE0381-C084 means failure of the requirement OE0381	
18. Verify in the source code that the stateChangeFrom field stores the value of the usageState attribute before the state change device when the deallocateCapacity operation changes the usageState.	<p><b>Pass:</b> The <i>stateChangeFrom</i> field stores the value of the usageState attribute before the state change. (OE0381-C085)</p> <p><b>Fail:</b> The <i>stateChangeFrom</i> field does not store the value of the usageState attribute before the state change. (OE0381-C085)</p>		Failure of this criterion OE0381-C085 means failure of the requirement OE0381	
19. Verify in the source code that the stateChangeTo field stores the value of the usageState attribute after the state change device when the deallocateCapacity operation changes the usageState.	<p><b>Pass:</b> The <i>stateChangeTo</i> field stores the value of the usageState attribute after the state change. (OE0381-C086)</p> <p><b>Fail:</b> The <i>stateChangeTo</i> field does not store the value of the usageState attribute after the state change. (OE0381-C086)</p>		Failure of this criterion OE0381-C086 means failure of the requirement OE0381	
<b>End of Test</b>				



Test Recording Log – OE_TC_079													
<b>Step1</b>	Name of device												
<b>Step2</b>	Source Code file of device.												
<b>Steps 3&amp;4</b>	<i>usageState</i> exists as read-only attribute?												
<b>Step5</b>	allocateCapacity/deallocateCapacity operations exist?												
<b>Step6</b>	Event sent by <i>allocateCapacity</i> op?						<b>Step13</b>	Event sent by <i>deallocateCapacity</i> op?					
<b>Step7</b>	IDM_CHANNEL used?						<b>Step14</b>	IDM_CHANNEL used?					
<b>Step8</b>	Producer ID is device identifier?						<b>Step15</b>	Producer ID is device identifier?					
<b>Step9</b>	Source ID is device identifier?						<b>Step16</b>	Source ID is device identifier?					
<b>Step10</b>	StateChange Category contains <i>usageState</i> Event?						<b>Step17</b>	StateChange Category contains <i>usageState</i> Event?					
<b>Step11</b>	StateChangeFrom field contains previous value?						<b>Step18</b>	StateChangeFrom field contains previous value?					
<b>Step12</b>	StateChangeTo field contains new value?						<b>Step19</b>	StateChangeTo field contains new value?					



## Test Summary OE\_TC\_079

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0345 \_\_\_\_\_

OE0381 \_\_\_\_\_

OE0381-C082 \_\_\_\_\_

OE0381-C083 \_\_\_\_\_

OE0381-C084 \_\_\_\_\_

OE0381-C085 \_\_\_\_\_

OE0381-C086 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

## B.3.12. OE\_TC\_080 - Device :: adminState commanded to be LOCKED

### Test Case Number: OE\_TC\_080

Device::adminState attribute

## Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0386	The adminState attribute shall contain the device's admin state value.
OE0387	The adminState attribute shall only allow the setting of LOCKED and UNLOCKED values, where setting "LOCKED" is only effective when the adminState attribute value is UNLOCKED, and setting "UNLOCKED" is only effective when the adminState attribute value is LOCKED or SHUTTING_DOWN.
OE0388	The adminState attribute, upon being commanded to be LOCKED, shall transition from the UNLOCKED to the SHUTTING_DOWN state and set the adminState to LOCKED for its entire aggregation of devices (if it has any).
OE0389	The adminState shall then transition to the LOCKED state when the device's usageState is IDLE and its entire aggregation of devices are LOCKED.

## References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2, 15 May 2006	Pages 3-60 – 3-61, Section 3.1.3.3.1.4.2; Page 3-59, Section 3.1.3.3.1.3.3; Page 3-62, Section 3.1.3.3.1.4.6; Page 3-64, Section 3.1.3.3.1.5.3.3; Page 3-74, Section 3.1.3.3.4.1
SCA Appendix C: Core Framework IDL	Version 2.2.2, 15 May 2006	Page C-27
SCA Appendix D: Domain Profile	Version 2.2.2, 15 May 2006	Page D-17, Section D.3.1.4.6

## Test Objective

This test case verifies requirements OE0386, OE0387, OE0388 and OE0389. The objective of this test case is to verify:

- that the adminState attribute contains a device's admin state (OE0386),
- that the adminState attribute may only be set to LOCKED or UNLOCKED, and
- that setting the adminState attribute to its current value (LOCKED or UNLOCKED) does not cause any processing to occur (OE0387).

This test case also verifies that the *adminState* attribute (of the device), upon being commanded to be LOCKED, transitions from the UNLOCKED to the SHUTTING\_DOWN state and sets the adminState to LOCKED for all its aggregated (i.e. children) devices (OE0388). Furthermore, this test case verifies that the adminState of an aggregate (i.e. parent) device transitions to the LOCKED state only after all of its aggregated devices are LOCKED and its usageState is IDLE (OE0389). If the device is not an aggregate device, then this test case must ensure that the usageState is IDLE before the adminState transitions into the LOCKED state (OE0389).

## Places to Verify

Devices

## IDL References

### Data

```
enum AdminType { LOCKED, SHUTTING_DOWN, UNLOCKED };
```

```
enum UsageType { IDLE, ACTIVE, BUSY };
```

```
attribute CF::Device::AdminType adminState;
```

```
readonly attribute CF::Device::UsageType usageState;
```

## Preconditions

- The Domain Profile files are available
- The source code of all Devices is available.

## Test Description

A. Identify and locate the devices of the Operating Environment. (OE0386, OE0387, OE0388, OE0389)

1. **N/A:** The Operating Environment does not contain devices.
2. **Untested:** The source code of the devices is not available.

For each device located in the Operating Environment, perform the following steps:

B. Verify that the attribute named “adminState” contains the device’s admin state value. (OE0386)

1. **Pass:** The attribute named “adminState” contains the device’s admin state value.
2. **Fail:** The attribute named “adminState” is not of type AdminType.
3. **Fail:** The attribute named “adminState” does not contain the device’s admin state.

C. Verify that the adminState attribute may only be set to LOCKED or UNLOCKED.

1. When setting the adminState attribute to LOCKED it is only effective when the adminState value is UNLOCKED. (OE0387)
  - a. **Pass:** The process of updating the adminState attribute to LOCKED includes a verification that the current adminState attribute value is UNLOCKED.
  - b. **Fail:** The process of updating the adminState attribute to LOCKED does not include a verification that the current adminState attribute value is UNLOCKED.
2. When setting the adminState attribute to UNLOCKED, it is only effective when the adminState value is LOCKED or SHUTTING\_DOWN. (OE0387)

- a. **Pass:** The *adminState* attribute update processing verifies that the current value is LOCKED or SHUTTING\_DOWN before beginning the processing associated with the UNLOCKED value.
    - b. **Fail:** The *adminState* attribute update processing does not verify that the current value is LOCKED or SHUTTING\_DOWN before beginning the processing associated with the UNLOCKED value.
  3. The *adminState* attribute cannot have its value set to SHUTTING\_DOWN. (OE0387)
    - a. **Pass:** The *adminState* attribute update processing has no processing associated with updating the admin state value with the SHUTTING\_DOWN value.
    - b. **Fail:** The *adminState* attribute update processing contains processing associated with updating the admin state with the SHUTTING\_DOWN value.
  - D. Identify the aggregate (i.e. parent) devices of the Core Framework, in other words, those devices that contain references to aggregated devices. (OE0388, OE0389)
    1. **N/A:** Test Descriptions steps E and F (the next two steps) are not applicable for a Core Framework that does not have aggregate (i.e., parent) devices. They may be skipped.
- For each aggregate (parent) device of the Core Framework, perform the following steps.
- E. Verify that when the parent device's *adminState* is commanded to be LOCKED, the following conditions occur:
    1. Verify that the parent device's *adminState* transitions from the UNLOCKED to the SHUTTING\_DOWN state. (OE0388)
      - a. **Pass:** The parent device's *adminState* transitions from the UNLOCKED to the SHUTTING\_DOWN state.
      - b. **Fail:** The parent device's *adminState* does not transition from the UNLOCKED to the SHUTTING\_DOWN state.
    2. Verify that the parent device sets the *adminState* of all of its child devices to the LOCKED state. (OE0388)
      - a. **Pass:** The parent device sets all of its child devices to the LOCKED state.
      - b. **Fail:** The parent device does not set all of its child devices to the LOCKED state.
  - F. Verify that the *adminState* of the parent device transitions into the LOCKED state only when its *usageState* is in the IDLE state and all of its child devices are LOCKED. (OE0389)
    1. **Pass:** The *adminState* of the parent device transitions into the LOCKED state only when all of its child devices are LOCKED and the *usageState* is in the IDLE state.
    2. **Fail:** The *adminState* transitions to the LOCKED state but its *usageState* is not set to IDLE.
    3. **Fail:** The *adminState* transitions to the LOCKED state when one or more of its child-aggregated devices (if they exist) are not set to LOCKED.
    4. **Fail:** The *adminState* does not transition into the LOCKED state when all of its aggregated devices are LOCKED and the *usageState* is IDLE.

For each device of the Core Framework that is not a parent device, perform the following steps:

- G. Verify that the *adminState* of the device transitions from the UNLOCKED state to the SHUTTING\_DOWN state if its *usageState* is not IDLE; If the *usageState* is already IDLE, it is allowable for the *adminState* to immediately transition to LOCKED. (OE0388)
1. **Pass:** The *adminState* transitions from the UNLOCKED state to the SHUTTING\_DOWN state when its *usageState* is not IDLE.
  2. **Fail:** The *adminState* state does not transition to the SHUTTING\_DOWN state when the *usageState* is not IDLE.
- H. Verify that the *adminState* of the device transitions into the LOCKED state only when its *usageState* is IDLE. (OE0389)
1. **Pass:** The *adminState* transitions to the LOCKED state only when its *usageState* is in the IDLE state.
  2. **Fail:** The *adminState* transitions to the LOCKED state but its *usageState* is not set to IDLE.

## Semi-automated Test Steps

After running JTAP's Device adminState attribute test, perform manual steps 9 through 14.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_080				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify and locate the devices of the Operating Environment (OE0386, OE0387, OE0388, and OE0389).</b>				
1. Examine the source code or Domain Profile files of the Core Framework and identify the names of the devices of the Core Framework.	<b>N/A:</b> The Core Framework does not contain devices. (OE0386, OE0387, OE0388, OE0389)		The names of the devices may be similar to the <i>usagename</i> 's in the DCD files. Examination of the Domain Profile to locate the devices may be done through using a semi-automated tool (see OE_TC_119)	
2. Locate the source code for each device and record the name of the file.	<b>Untested:</b> The source code of the devices is not located. (OE0386, OE0387, OE0388, OE0389)		Search the source code for the names of the devices and determine the source code file for each device. One possible way to do this is to check that one or more of the SCA Device interfaces are implemented in the source code of the component ( <i>Device</i> , <i>ExecutableDevice</i> , <i>LoadableDevice</i> )	
<b>For each device located in the Operating Environment, perform the following steps:</b>				
<b>B. Verify that the attribute named "adminState" contains the device's admin state value. (OE0386)</b>				
3. Verify that there is an attribute named "adminState" that is of type AdminType.	<b>Pass:</b> An AdminType attribute named "adminState" exists. (OE0386)  <b>Fail:</b> An AdminType attribute named "adminState" does not exist. (OE0386)			

OE_TC_080				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Locate methods that update the adminState attribute.	<b>Pass:</b> Methods for updating the adminState attribute are found. (OE0386)  <b>Fail:</b> No methods were found that update the adminState attribute. (OE0386)			
5. Verify that the attribute named "adminState" contains the device's administration state.	<b>Pass:</b> The attribute named "adminState" contains the device's administration state value. (OE0386)  <b>Fail:</b> The attribute named "adminState" does not contain the device's administration state value. (OE0386)			
<b>C. Verify that the adminState attribute may only be set to LOCKED or UNLOCKED.</b>				
<b>1. When setting the adminState attribute to LOCKED it is only effective when the adminState value is UNLOCKED. (OE0387)</b>				
6. Locate the processing for updating the "adminState" attribute.	<b>Pass:</b> The processing for updating the "adminState" attribute is found. (OE0387)  <b>Fail:</b> The processing for updating the "adminState" attribute is not found. (OE0387)		This could be called the setter method for the "adminState" attribute.	
7. Verify that when there is an attempt to set the "adminState" attribute to LOCKED, the update processing verifies that the current admin state is	<b>Pass:</b> The process of updating the adminState attribute to LOCKED includes a verification that the current adminState attribute value is UNLOCKED. (OE0387)			

OE_TC_080				
Steps	Expected Results	Actual Results	Comments	Test Result
UNLOCKED before proceeding.	<b>Fail:</b> The process of updating the adminState attribute to LOCKED does not include a verification that the current adminState attribute value is UNLOCKED. (OE0387)			
<b>2. When setting the adminState attribute to UNLOCKED, it is only effective when the adminState value is LOCKED or SHUTTING_DOWN. (OE0387)</b>				
8. Verify that when there is an attempt to set the “adminState” attribute to LOCKED, the current state may not be LOCKED or SHUTTING_DOWN.	<p><b>Pass:</b> The update processing verifies that the current admin state is LOCKED, before proceeding with the processing associated with setting it to UNLOCKED. (OE0387)</p> <p><b>Fail:</b> The update processing does not verify that the current admin state is LOCKED, before proceeding with the processing associated with setting it to UNLOCKED. (OE0387)</p>			
<b>3. The adminState attribute cannot have its value set to SHUTTING_DOWN. (OE0387)</b>				
9. Verify that when there is an attempt to set the “adminState” attribute to SHUTTING_DOWN, no processing occurs.	<p><b>Pass:</b> The update processing does nothing when attempting to set the admin state to SHUTTING_DOWN. (OE0387)</p> <p><b>Fail:</b> The update processing does some processing when attempting to set the admin</p>			



OE_TC_080				
Steps	Expected Results	Actual Results	Comments	Test Result
	state to SHUTTING_DOWN. (OE0387)			
<b>D. Identify the aggregate (i.e. parent) devices of the Core Framework, in other words, those devices that contain references to aggregated devices. (OE0388, OE0389)</b>				
10. Identify the aggregate (i.e., parent) devices of the Core Framework by examining the source code and Domain Profile files.	N/A: Test descriptions C and D are not applicable for a Core Framework that does not have aggregate (i.e., parent) devices. (OE0388, OE0389)		<p>One possible way to find the parent devices is to look at the Domain Profile XML(DCD, SCD) and determine if any of the provides ports of the device contains clues that the device has children (i.e. look for such strings as aggregate, composite, child...)</p> <p>The DPD of a parent device may also contain references to the <i>childhwdevice</i> element (page D-17, SCA v2.2.2, Appendix D) or a child device may refer to its parent device by the <i>compositepartofdevice</i> element in the DCD.</p> <p>Using source code examination methods, search the device for references to the <i>Aggregate</i> device interface. (SCA v2.2.2, page 3-74)</p> <p>If the <i>compositeDevice</i> attribute of the device is not nil, then this is also good indication that device has children (SCA v2.2.2, page 3-62)</p>	
<b>For each aggregate (parent) device of the Core Framework, perform the steps C and D.</b>				
<b>E. Verify that when the parent device's <i>adminState</i> is commanded to be LOCKED, the following conditions occur: (OE0388)</b>				
<b>1. Verify that the parent device's <i>adminState</i> transitions from the UNLOCKED to the SHUTTING_DOWN state.</b> <b>2. Verify that the parent device sets the <i>adminState</i> of all of its child devices to the LOCKED state.</b>				
11. Identify the location in the parent device's source code where the <i>adminState</i> attribute is commanded to be LOCKED.	The location where the <i>adminState</i> attribute is commanded to be LOCKED is identified.		The location is mostly likely a "setter" function (i.e. <i>setAdminState</i> ) or could possibly be in or around the <i>deallocateCapacity</i> or <i>releaseObject</i> operations.	

OE_TC_080				
Steps	Expected Results	Actual Results	Comments	Test Result
12. Verify in the source code identified in step 4 that the device's <i>adminState</i> transitions from the UNLOCKED to the SHUTTING_DOWN state.	<p><b>Pass:</b> The <i>adminState</i> transitions from the UNLOCKED to the SHUTTING_DOWN state. (OE0388)</p> <p><b>Fail:</b> The <i>adminState</i> does not transition from the UNLOCKED to the SHUTTING_DOWN state. (OE0388)</p>			
13. Verify in the source code identified in step 4 that the device sets all of its child devices to the LOCKED state when the parent device itself is commanded to be LOCKED.	<p><b>Pass:</b> The <i>adminState</i> sets all of its child devices to the LOCKED state. (OE0388)</p> <p><b>Fail:</b> The <i>adminState</i> does not set one or more of its child devices to the LOCKED state. (OE0388)</p>			
<b>F. Verify that the <i>adminState</i> of the parent device transitions into the LOCKED state only when its <i>usageState</i> is in the IDLE state and all of its child devices are LOCKED. (OE0389)</b>				
14. Verify that logic exists in the parent device's source code to transition into the LOCKED state only when all of its aggregated devices are set to LOCKED and the <i>usageState</i> is IDLE.	<p><b>Pass:</b> The <i>adminState</i> of the parent device successfully transitions into the LOCKED state when all of its aggregated devices are LOCKED and its <i>usageState</i> is set to IDLE. (OE0389)</p> <p><b>Fail:</b> The <i>adminState</i> transitions to LOCKED but one or more of its aggregated devices are not LOCKED. (OE0389)</p>			

OE_TC_080				
Steps	Expected Results	Actual Results	Comments	Test Result
	<p><b>Fail:</b> The <i>adminState</i> transitions to LOCKED but the <i>usageState</i> is not IDLE. (OE0389)</p> <p><b>Fail:</b> The <i>adminState</i> does not transition into the LOCKED state when all of its aggregated devices are LOCKED and the <i>usageState</i> is IDLE. (OE0389)</p>			
For each device of the Core Framework that is not a parent device, perform the following steps.				
<b>G. Verify that the <i>adminState</i> of the device transitions from the UNLOCKED state to the SHUTTING_DOWN state if its <i>usageState</i> is not IDLE; If the <i>usageState</i> is already IDLE, it is allowable for the <i>adminState</i> to immediately transition to LOCKED. (OE0388)</b>				
15. Identify the location in the device source code where the <i>adminState</i> attribute is set.	The location where the <i>adminState</i> attribute is set LOCKED is identified.		The location is mostly likely a “setter” function (i.e. <i>setAdminState</i> ) or could possibly be in or around the <i>deallocateCapacity</i> or <i>releaseObject</i> operations.	
16. Verify in the source code found in step 8 that the <i>adminState</i> of the device transitions from the UNLOCKED state to the SHUTTING_DOWN state if its <i>usageState</i> is not IDLE.	<p><b>Pass:</b> The <i>adminState</i> transitions from the UNLOCKED state to the SHUTTING_DOWN state if the <i>usageState</i> is not IDLE. (OE0388)</p> <p><b>Fail:</b> The <i>adminState</i> state does not transition to the SHUTTING_DOWN state if the <i>usageState</i> is not IDLE. (OE0388)</p>		It is acceptable for the <i>adminState</i> to immediately fall through from the SHUTTING_DOWN state to the LOCKED state if the <i>usageState</i> is already ensured to be in the IDLE state.	
<b>H. Verify that the <i>adminState</i> of the non-parent device transitions into the LOCKED state only when its <i>usageState</i> is IDLE. (OE0389)</b>				
17. Perform this step if the device is not a parent. Verify	1. <b>Pass:</b> The device's <i>adminState</i> only transitions to			

OE_TC_080				
Steps	Expected Results	Actual Results	Comments	Test Result
in the source code that the <i>adminState</i> is only allowed to transition to the LOCKED state when its <i>usageState</i> is IDLE.	LOCKED when its <i>usageState</i> is IDLE. (OE0389)  2. <b>Fail:</b> The device's <i>adminState</i> is allowed to transition to the LOCKED state when its <i>usageState</i> is not in the IDLE state. (OE0389)			
End of Test				

Test Recording Log – OE_TC_080								
Step1 (device names)								
Step2 (file names)	Steps 3 - 5 adminState attribute	Steps 6 - 9 (setting adminState with LOCKED & UNLOCKED)	Step10 (parent?)	Perform these steps only if the device is a parent.			Non-Parent devices	
				Step12 (SHUTTING _DOWN state?)	Step13 (sets child devices to LOCKED?)	Step14 (IDLE and child devices LOCKED?)	Steps 15 & 16 (SHUTTING _DOWN state if not IDLE?)	Step17 (IDLE before LOCKED?)

## Test Summary OE\_TC\_080

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0386 \_\_\_\_\_

OE0387 \_\_\_\_\_

OE0388 \_\_\_\_\_

OE0389 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.13. OE\_TC\_081 - Device :: allocateCapacity****Test Case Number:** OE\_TC\_081

Device::allocateCapacity

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0405	The <i>allocateCapacity</i> operation shall reduce the current capacities of the device based upon the input capacities parameter, when the device's adminState is UNLOCKED, device's operationalState is ENABLED, and device's usageState is not BUSY.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2, 15 May 2006	Page 3-62, Section 3.1.3.3.1.5.1.3.
SCA, AppendixD: Domain Profile	Version 2.2.2, 15 May 2006	Page D-13, Section D. 2.1.7; Page D-22, Section D.4.1.1.6

**Test Objective**

This test case verifies requirement, OE0405. The objective of this test case is to verify that the *allocateCapacity* operation reduces the capacities of the device using the allocation capacities specified in the input parameter. The properties for the allocation capacities should be defined in the XML. The *allocateCapacity* operation should always check that the device's adminState is UNLOCKED, the device's operationalState is ENABLED, and the device's usageState is not BUSY before it reduces the current capacities of the device.

**Places to Verify**

Devices

**IDL References****Data**

enum AdminType { LOCKED, SHUTTING\_DOWN, UNLOCKED };

enum OperationalType { ENABLED, DISABLED };

enum UsageType { IDLE, ACTIVE, BUSY };

## Operations

boolean *allocateCapacity* (in Properties capacities) raises (InvalidCapacity, InvalidState);

## Preconditions

- Domain Profile test is passed
- All Domain Profile files are available
- The source code files of the Operating Environment are available.

## Test Description

- A. Identify the devices of the OE. (OE0405)
  - a. **Pass:** The devices of the OE are identified and their source code and Domain Profile(DCD and SPD) files are located.
  - b. **Untested:** The OE contains devices but their source code or Domain Profile files (DCD or SPD) do not exist.
  - c. **N/A:** The OE does not contain devices.
- B. Identify the capacity model that the device's *allocateCapacity* operation uses. (OE0405)
  - a. **Pass:** The capacity model of the device's *allocateCapacity* operation is identified.
  - b. **N/A:** The capacity model of the device's *allocateCapacity* operation is not identified.

**Note:** A capacity model is the usage of allocation capacities for certain device functionality. The allocation properties are defined in the device's software profile (XML).
- C. Verify that the *allocateCapacity* operation exists and is only allowed to reduce the capacities of the device when the device's adminState is UNLOCKED, the device's operationalState is ENABLED, and the device's usageState is not BUSY and these state conditions all occur simultaneously. (OE0405)
  1. **Pass:** The *allocateCapacity* operation exists and reduces the capacities of the device only when all three of the following state conditions occur simultaneously: the device's AdminState is UNLOCKED, the device's operationalState is ENABLED, and the device's usageState is not BUSY.
  2. **Fail:** The *allocateCapacity* operation does not exist or if it does, reduces the capacities of the device even though all three of the following state conditions are not fulfilled at the same time: the device's AdminState is UNLOCKED, the device's operationalState is ENABLED, and the device's usageState is not BUSY.
- D. Verify that the *allocateCapacity* operation uses the allocation properties defined in its parameter to reduce the capacities of the device. (OE0405)
  1. **Pass:** The *allocateCapacity* operation uses the allocation properties defined in its parameter to reduce the capacity of the device when executed.



2. **Fail:** The *allocateCapacity* operation does not use the allocation properties defined in the parameter to reduce the capacity of the device when executed.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_081				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the devices of the Operating Environment. (OE0405)</b>				
1. Examine the DCD file(s) for each device locate the name and SPD file of each device. Record a list of the name and SPD file of each device.	<p><b>Pass:</b> The names and SPD files of the devices are obtained by the Domain Profile/DCD files. (OE0405)</p> <p><b>N/A:</b> The names of the devices are not available. (OE0405)</p> <p><b>Untested:</b> The OE contains devices but their Domain Profile files (DCD, SPD) cannot be located. (OE0405)</p>		<p>A device's id and name are usually specified in the DCD file as in the following example:</p> <pre>&lt;componentplacement&gt;   &lt;componentfileref refid="xxxx " /&gt;   &lt;componentinstantiation     id="DCE:xxxx-yyyyy-zzzz"&gt;     &lt;usagename&gt;zzzz&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt; &lt;componentfiles&gt; &lt;componentfile id="xxxx " type="SPD"&gt;   &lt;localfile     name=".../logservice/yyyyy.spd.xml"/&gt;   &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre>	
2. Locate the source code for the devices and record the name of the files in a list.	<p><b>Pass:</b> The source code for the devices is located. (OE0405)</p> <p><b>Untested:</b> The source code for the devices is not located. (OE0405)</p>		Using the clue of the component placement ref and the information found in the SPD file could also give additional information on how to locate the source code.	
<b>B. Identify the capacity model that the device's <i>allocateCapacity</i> operation uses. (OE0405)</b> <b>Note:</b> A capacity model is the usage of allocation capacities for certain device functionality. The allocation properties are defined in the device's software profile (XML).				

OE_TC_081				
Steps	Expected Results	Actual Results	Comments	Test Result
<p>3. Using appropriate source code, XML and developer documentation, identify the device's allocation properties, which are determined by the capacity model. List the allocation properties. (OE0405)</p> <p><b>Note:</b> This may include determining the maximum capacities for each allocation property of the device(s)</p>	<p><b>Pass:</b> The allocation properties of the device are identified. (OE0405)</p> <p><b>N/A:</b> The allocation properties of the device are not identified. (OE0405)</p>		<p>SCA v2.2.2 pg. 3-58</p> <p>Capacity Operations - In order to use a device, certain capacities (e.g., memory, performance, etc.) are obtained from the device. A device may have multiple capacities, which need to be allocated, since each device has its own unique capacity model, which is described in the associated software profile.</p> <p>The following is a possible example of an allocation for a device inside a properties XML file.</p> <pre>&lt;/simplesequence&gt;   &lt;simple id="DCE:xxxxx-yyyyy- zzzzz" type="ulong" name="XXXXXX" mode="readwrite"&gt;&lt;description&gt;Modem Device has allocation ....&lt;/description&gt;   &lt;value&gt;xx&lt;/value&gt;   &lt;kind kindtype="allocation"/&gt; &lt;action type="xx"/&gt;&lt;/simple&gt;</pre>	
<p><b>C. Verify that the <i>allocateCapacity</i> operation exists and is only allowed to reduce the capacities of the device when the device's adminState is UNLOCKED, the device's operationalState is ENABLED, and the device's usageState is not BUSY and these state conditions all occur simultaneously. (OE0405)</b></p>				
<p>4. Verify that the <i>allocateCapacity</i> operation exists.</p>	<p><b>Pass:</b> The <i>allocateCapacity</i> operation exists. (OE0405)</p> <p><b>Fail:</b> The <i>allocateCapacity</i> operation does not exist. (OE0405)</p>			

OE_TC_081				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify in the source code that the <i>allocateCapacity</i> operation is only allowed to reduce the capacities of the device when all three state conditions occur simultaneously: The device's operationalState is ENABLED, the device's adminState is UNLOCKED and the device's usageState is not BUSY.	<p><b>Pass:</b> The <i>allocateCapacity</i> operation reduces the capacities of the device only when all three of the following state conditions occur simultaneously: the device's AdminState is UNLOCKED, the device's operationalState is ENABLED, and the device's usageState is not BUSY. (OE0405)</p> <p><b>Fail:</b> The <i>allocateCapacity</i> operation reduces the capacities of the device even though all three of the following state conditions are not fulfilled at the same time: the device's AdminState is UNLOCKED, the device's operationalState is ENABLED, and the device's usageState is not BUSY. (OE0405)</p>			
<b>D. Verify that the <i>allocateCapacity</i> operation uses the allocation properties defined in its parameter to reduce the capacities of the device. (OE0405)</b>				

OE_TC_081				
Steps	Expected Results	Actual Results	Comments	Test Result
6. Verify in the source code that the <i>allocateCapacity</i> operation uses the allocation properties defined in its parameter to reduce the capacity of the device.	<b>Fail:</b> The <i>allocateCapacity</i> operation does not use the allocation properties defined in the parameter to reduce the capacity of the device when executed. (OE0405)  <b>Pass:</b> The <i>allocateCapacity</i> operation uses the allocation properties defined in its parameter to reduce the capacity of the device when executed. (OE0405)		The following is an example of what an allocation property may look like inside of a PRF file.  <simple id="DCE:xxxxx-yyyyy-zzzzz" type="ulong" name="XXXX" mode="readwrite"> <description>Modem Device has allocation property.</description> <value>xxx</value> <kind kindtype="allocation"/><action type="xx"/></simple>	
End of Test				

Test Recording Log - OE_TC_081					
Step 1 (Devices(number, names, etc.) of the Operating Environment are determined.(e.g. through XML, source)	Step 2 (Source code for devices is located.)	Step 3 List of allocation property ids.	Step 4 (allocateCapacity operation exists.)	Step 5 (Three states occur simultaneously: adminState is UNLOCKED, operationalState is ENABLED, usageState is BUSY, when allocateCapacity reduces capacities of device.)	Step 6 (allocateCapacity operation reduces capacities in device using parameter values)

## Test Summary OE\_TC\_081

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0405 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.14. OE\_TC\_082 - Device :: allocateCapacity BUSY****Test Case Number:** OE\_TC\_082

Device::allocateCapacity

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0406	The <i>allocateCapacity</i> operation shall set the Device's usageState attribute to BUSY, when the device determines that it is not possible to allocate any further capacity.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-58, Section 3.1.3.3.1.5.1.3; page 3-62; Section 3.1.3.3.1.3.5; Section 3.1.3.3.1.4.1
SCA Appendix D: Domain Profile	Version 2.2.2, 15 May 2006	Page D-13, Section D 2.1.7.

**Test Objective**

This test case verifies OE0406. The objective of this test is to verify that the *allocateCapacity* operation of the device can accurately determine when it is not able to allocate any further capacity. Capacities for a device are usually specified by properties referenced in the DCD and SPD files. The *allocateCapacity* operation must set the usageState attribute to BUSY when the *allocateCapacity* operation determines that no further capacity can be allocated.

**Places to Verify**

Devices

**IDL References****Data**

UsageType { IDLE, ACTIVE, BUSY };

**Exceptions**boolean *allocateCapacity* (in Properties capacities) raises (InvalidCapacity, InvalidState);



## Preconditions

- The Domain Profile files, including the DCD file(s), are available.
- The Domain Profile test is passed.
- The source code files of the Core Framework and devices are available.

## Test Description

### A. Identify the devices of the Operating Environment. (OE0406)

1. **Pass:** The devices of the OE are identified.
2. **N/A:** The OE does not contain devices.
3. **Untested:** The OE contains devices but their Domain Profile files or source code is not available.

For each device in the Operating Environment:

### B. Verify that the *allocateCapacity* operation exists and the *usageState* attribute exists for the device. (OE0406)

1. **Pass:** The *allocateCapacity* operation exists and the device contains the *usageState* attribute.
2. **Fail:** The *allocateCapacity* operation does not exist.
3. **Fail:** The device does not contain the *usageState* attribute.

### C. Identify the capacity model that the device's *allocateCapacity* operation uses. (OE0406)

1. **Pass:** The capacity model of the device's *allocateCapacity* operation is identified.
2. **N/A:** The capacity model of the device's *allocateCapacity* operation is not identified.

**Note:** A capacity model is the usage of allocation capacities for certain device functionality; the allocation properties are defined in the device's software profile (XML).

### D. Verify that the device's *allocateCapacity* operation can determine the threshold level at which the maximum capacity of the device has been met and that it sets the *usageState* to BUSY when the device reaches this capacity. (OE0406)

1. Verify that the threshold level of capacity for the device can be determined by the *allocateCapacity* operation.
  - a. **Pass:** The threshold level of capacity for the device can be determined by the *allocateCapacity* operation
  - b. **Fail:** The threshold level of capacity for the device cannot be determined by the *allocateCapacity* operation
2. Verify that the *allocateCapacity* operation sets the *usageState* of the device to BUSY when it is determined that it is not possible to allocate capacity further on the device.
  - a. **Pass:** The *allocateCapacity* operation sets the *usageState* of the device to BUSY when it is determined that it is not possible to allocate capacity further on the device.
  - b. **Fail:** The *allocateCapacity* operation does not set the *usageState* of the device to BUSY when it is determined that it is not possible to allocate capacity further on the device.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_082				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the devices of the Operating Environment. (OE0406)</b>				
1. Examine the DCD files to identify the devices provided by the OE and record the SPD file associated with each device.	<p><b>Pass:</b> The DCD file(s) indicate the names of the devices and the associated SPD files. (OE0406)</p> <p><b>N/A:</b> The OE does not contain devices. The test ends. (OE0406)</p> <p><b>Untested:</b> The OE contains devices but either at least one DCD does not exist or the SPD files for any of the devices do not exist. The test ends. (OE0406)</p>		<p>A device's id and name are usually specified in the DCD file as in the following example:</p> <pre>&lt;componentplacement&gt;   &lt;componentfileref refid="CDATA"&gt;     &lt;componentinstantiation       id="DCE:ddd123xxxx"&gt;       &lt;usagename&gt;aDevice&lt;/usagename&gt;     &lt;/componentinstantiation&gt;   &lt;/componentplacement&gt; &lt;/componentfiles&gt; &lt;componentfile id="&lt;CDATA&gt;"   type="SPD"&gt;   &lt;localfile     name=".../logservice/DeviceFile.spd.xml"/&gt;   &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre>	
2. Locate the source code for the devices and record the names of the files.	<p><b>Untested:</b> The source code of the devices is not located. The test ends. (OE0406)</p>		<p>The development engineer and the developer documentation, if available, is the best source for the location of the source code.</p> <p>In some cases, it is possible to trace the location of the source code from the executable declaration in the SPD.</p>	
<p><b>For each device in the Operating Environment:</b></p> <p><b>B. Verify that the <i>allocateCapacity</i> operation exists and the <i>usageState</i> attribute exists for the device. (OE0406)</b></p>				

OE_TC_082				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify in the source code that the <i>allocateCapacity</i> operation exists for the device.	<b>Pass:</b> The <i>allocateCapacity</i> operation exists. (OE0406)  <b>Fail:</b> The <i>allocateCapacity</i> operation does not exist. (OE0406)		There is usually an <i>allocateCapacity</i> operation for each device. In some cases, <i>allocateCapacity</i> might be declared in other parts of the OE and it should be determined in these implementations affect the outcome of the requirement for the specific device.	
4. Verify in the source code that the device's <i>usageState</i> attribute exists.	<b>Pass:</b> The device contains the <i>usageState</i> attribute. (OE0406)  <b>Fail:</b> The device does not contain the <i>usageState</i> attribute. (OE0406)			
<b>C. Identify the capacity model that the device's <i>allocateCapacity</i> operation uses. (OE0406)</b> <b>Note:</b> A capacity model is the usage of allocation capacities for certain device functionality; The allocation properties are defined in the device's software profile (XML).				

OE_TC_082				
Steps	Expected Results	Actual Results	Comments	Test Result
<p>5. Using appropriate source code, XML and developer documentation, identify the capacity model for the device's <i>allocateCapacity</i> operation.</p> <p><b>Note:</b> This may include determining the maximum capacities for each allocation property of the device(s).</p>	<p><b>N/A:</b> The capacity model of the device's <i>allocateCapacity</i> operation is not determined. (OE0406)</p>		<p>SCA v2.2.2 pg. 3-58</p> <p>Capacity Operations - In order to use a device, certain capacities (e.g., memory, performance, etc.) are obtained from the device. A device may have multiple capacities, which need to be allocated, since each device has its own unique capacity model, which is described in the associated software profile.</p> <p>The following is a possible example of an allocation for a device inside a properties XML file.</p> <pre> &lt;/simplesequence&gt;   &lt;simple id="DCE:12aaaaa" type="ulong" name="MDDDT_TX" mode="readwrite"&gt;&lt;description&gt;Modem Device has allocation ....&lt;/description&gt;   &lt;value&gt;1&lt;/value&gt;   &lt;kind kindtype="allocation"/&gt; &lt;action type="ge"/&gt;&lt;/simple&gt; </pre>	
<p><b>D. Verify that the device's <i>allocateCapacity</i> operation can determine the threshold level at which the maximum capacity of the device has been met and that it sets the usageState to BUSY when the device reaches this capacity. (OE0406)</b></p> <ol style="list-style-type: none"> <li>1. Verify that the threshold level of capacity for the device can be determined by the <i>allocateCapacity</i> operation.</li> <li>2. Verify that the <i>allocateCapacity</i> operation sets the usageState of the device to BUSY when it is determined that it is not possible to allocate capacity further on the device.</li> </ol>				

OE_TC_082				
Steps	Expected Results	Actual Results	Comments	Test Result
6. Verify in the source code that logic exists for the <i>allocateCapacity</i> operation to determine the threshold level at which it is not possible to allocate further capacity.	<b>Pass:</b> The threshold level of capacity for the device can be determined by the <i>allocateCapacity</i> operation. (OE0406)  <b>Fail:</b> The threshold level of capacity for the device cannot be determined by the <i>allocateCapacity</i> operation. (OE0406)			
7. Verify in the source code that the <i>allocateCapacity</i> operation sets the <i>usageState</i> to BUSY when it detects the threshold allocation capacity has been met.	<b>Pass:</b> The <i>allocateCapacity</i> operation sets the <i>usageState</i> of the device to BUSY when a determination is made that it is not possible to allocate capacity further on the device. (OE0406)  <b>Fail:</b> The <i>allocateCapacity</i> operation does not set the <i>usageState</i> of the device to BUSY when it is determined that it is not possible to allocate capacity further on the device. (OE0406)			
<b>End of Test</b>				

Test Recording Log – OE_TC_082						
Step1 (Determine the names and SPD files for each device.)	Step2 (Source code file for each device.)	Step3 ( <i>allocateCapacity</i> operation exists.)	Step4 (usageState for device exists)	Step5 (Identify capacity model of device.)	Step6 (Logic exists to determine threshold level of device where it cannot further allocate capacity)	Step7 <i>allocateCapacity</i> operation sets usageState to busy when device detects allocation threshold has been met.)

## Test Summary OE\_TC\_082

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0406\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.15. OE\_TC\_083 - Device :: allocateCapacity Failure****Test Case Number:** OE\_TC\_083

Device::allocateCapacity

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0408	The <i>allocateCapacity</i> operation shall return TRUE, if the capacities have been allocated, or FALSE, if not allocated.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-62 / section 3.1.3.3.1.5.1.4

**Test Objective**

This test case verifies OE0408. The objective is to determine whether an allocation request has the possibility of failing, if it can, and if it does, then verify that the requirement to return FALSE is met by the *allocateCapacity* operation.

NOTE: It is possible that an *allocateCapacity* operation might have no failure branch, therefore always returning TRUE, never FALSE. There are also two exception conditions {InvalidCapacity, InvalidState} that may be raised.

**Places to Verify**

Devices

**IDL References****Data**

```
struct DataType {  
    string id;  
    any value; };  
typedef sequence <DataType> Properties;
```

**Exceptions**

```
exception InvalidCapacity { string msg; Properties capacities; };  
exception InvalidState { string msg; };
```



## Operations

boolean *allocateCapacity* (in Properties capacities)  
    raises (InvalidCapacity, InvalidState);

## Preconditions

- All the Device Interface source code files are available.

## Test Description

- A. Determine if the *allocateCapacity* operation source code has checks to determine if capacity allocations (e.g. memory) are successful (OE0408).
  1. **Pass:** The *allocateCapacity* operation has no checks and returns only TRUE.  
    Note: if this is the case, skip the remaining steps of this test.
- B. Verify that an allocation failure will result in a FALSE return value (OE0408).  
    Note: The *allocateCapacity* operation probably tests for InvalidCapacity and InvalidState prior to attempting to allocate.
  1. **Pass:** The *allocateCapacity* returns a FALSE return value.
  2. **Fail:** The *allocateCapacity* raises an InvalidCapacity exception when an allocation failure occurs, i.e. in a place where it should return a FALSE return value.
  3. **Fail:** The *allocateCapacity* raises an InvalidState exception when an allocation failure occurs, i.e. in a place where it should return a FALSE return value.
  4. **Fail:** The *allocateCapacity* returns a TRUE return value when an allocation failure occurs.
- C. Verify that if all allocations are successful, the *allocateCapacity* operation will return a TRUE return value (OE0408).
  1. **Pass:** The *allocateCapacity* returns a TRUE return value.
  2. **Fail:** The *allocateCapacity* raises an InvalidCapacity exception when an allocation is successful, i.e. in a place where it should return a TRUE return value. **Fail:** The *allocateCapacity* raises an InvalidState exception when an allocation is successful, i.e. in a place where it should return a TRUE return value.
  3. **Fail:** The *allocateCapacity* returns an erroneous FALSE return value when an allocation is successful.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_083				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine if the <i>allocateCapacity</i> operations source code has checks to determine if capacity allocations (e.g. memory) are successful (OE0408).</b>				
1. Locate the <i>allocateCapacity</i> operation source code	Source code located			
2. Look for the implementation of requested allocations within the operation.	Some may be found. It is also possible that there is none.			
3. Determine if it is possible for any requested allocations to fail.	<b>Pass:</b> The <i>allocateCapacity</i> operation has no checks and returns only TRUE. (OE0408)			
	Note: if this is the case, skip the remaining steps of this test.			
<b>B. Verify that an allocation failure will result in a FALSE return value (OE0408).</b>				
4. Identify causes of the InvalidCapacity exception	<p>This exception should be due to errors in the capacities input parameter, not due to unavailability of a valid capacity request.</p> <p><b>Fail:</b> The <i>allocateCapacity</i> raises an InvalidCapacity exception in a place where it should return a FALSE return value. (OE0408)</p>			

OE_TC_083				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Identify causes of the InvalidState exception	<p>This exception should be due to errors in any OE/Device internal state, not due to unavailability of a valid capacity request.</p> <p><b>Fail:</b> The <i>allocateCapacity</i> raises an InvalidState exception in a place where it should return a FALSE return value. (OE0408)</p>			
6. Verify that the return value is used to indicate failure to allocate	<p>The failure to allocate required capacities will result in a FALSE return value.</p> <p><b>Fail:</b> The <i>allocateCapacity</i> returns an erroneous TRUE return value under conditions where it should return FALSE. (OE0408)</p> <p><b>Pass:</b> The <i>allocateCapacity</i> returns a FALSE return value under failure conditions. (OE0408)</p>			
<b>C. Verify that if all allocations are successful, the <i>allocateCapacity</i> operation will return a TRUE return value (OE0408).</b>				
7. Verify that the return value of TRUE is used to indicate a successful allocation.	<b>Pass:</b> The <i>allocateCapacity</i> returns a TRUE return value upon successful allocation of all required capacities. (OE0408)			
<b>End of Test</b>				

Test Recording Log – OE_TC_083				
Step1 <i>allocateCapacity</i> source code	Step4 InvalidCapacity exception (Y/N)	Step5 InvalidState exception (Y/N)	Step6 Allocation Failure returns FALSE (Y/N)	Step7 Allocation Success returns TRUE (Y/N)

## Test Summary – OE\_TC\_083

Once testing is complete for every component of the OE under test, report the test result as follows:

Pass: No failures detected  
Fail: Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
Untested: Condition which is not testable  
N/A: Not Applicable

**Overall Test Result** (Pass, Fail, Untested, or N/A): \_\_\_\_\_

OE0408 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.16. OE\_TC\_084 - Device :: deallocateCapacity****Test Case Number:** OE\_TC\_084

Device::deallocateCapacity

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0411	The deallocateCapacity operation shall adjust the current capacities of the device based upon the input capacities parameter.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2, 15 May 2006	Page 3-63 to 3-64, Section 3.1.3.3.1.5.2.3

**Test Objective**

This test case verifies requirement OE0411. The objective is to test that capacities are properly deallocated within the *deallocateCapacity()* operation.

**Places to Verify**

Devices

**IDL References****Data**

```
struct DataType {  
    string id;  
    any value; };  
typedef sequence <DataType> Properties;
```

**Exceptions**

```
exception InvalidCapacity { string msg; Properties capacities; };  
exception InvalidState { string msg; };
```

## Operations

void deallocateCapacity (in Properties capacities)  
    raises (InvalidCapacity, InvalidState);

## Preconditions

- The source code files are available.

## Test Description

A. Identify the devices of the Operating Environment. (OE0411)

For each of the devices in the Operating Environment:

B. Evaluate the *allocateCapacity()* and *deallocateCapacity()* operations.

1. Determine if the *allocateCapacity()* and *deallocateCapacity()* provide for shared uses of the device. Perform step 2 for capacities that do not support multiple allocations and perform step 3 for capacities that do support multiple allocations.
2. If the device does not support multiple allocations of any device capacity, then verify that the *deallocateCapacity()* operation returns the selected capacity to its respective full capacity. This is descriptive of the simplest form of device, every allocated capacity is either <allocated / not available> or <not allocated / available>. (OE0411)
  - a. **Pass:** The *deallocateCapacity()* operation restore specified capacities.
  - b. **Fail:** The *deallocateCapacity()* operation fails to restore specified capacities.
3. If the device supports multiple allocations of any capacity, then verify that the *allocateCapacity()* and *deallocateCapacity()* keeps track how much of said capacity is allocated and deallocated in each operational call. Accurate accounting must consider further complications to capacity adjustment; allocations and deallocations are normally coordinated in pairs; an unpaired deallocation operation should not create capacity in excess of the actual capacity of the device. (OE0411)
  - a. **Pass:** The *deallocateCapacity()* operation restores specified capacities by their prescribed quantities and to not more than their full quantity.
  - b. **Fail:** The *deallocateCapacity()* operation fails to restore specified capacities by their prescribed quantities.
  - c. **Fail:** The *deallocateCapacity()* operation allows specified capacities to be restored in excess of their original capacity.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_084				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the devices of the Operating Environment. (OE0411)</b>				
1. Locate the DCD file for each device and record the name and SPD file of each device.  <b>NOTE:</b> The best way to determine the names of the devices is to use the XML files. In some cases, though, developer documentation and conversations with the developer themselves may be a faster way to locate the names and SPD files of the device(s).	The names of the devices are obtained by the Domain Profile/DCD files.			
2. Locate the source code for the device and record the name of the file.	The source code for the devices is located.			
<b>For each of the devices in the Operating Environment:</b>				
<b>B. Evaluate the source code of the <i>allocateCapacity()</i> and <i>deallocateCapacity()</i> operations. (OE0411)</b>				
3. Locate the <i>deallocateCapacity()</i> operation source code; also locate the <i>allocateCapacity()</i> operation source code.	Source code located			
4. Determine if the <i>allocateCapacity()</i> and <i>deallocateCapacity()</i> provide for shared uses of the device. Perform step 5 for capacities that do not support multiple allocations and perform step 6 for capacities that do support multiple allocations.	For a device with multiple capacities, some may be sharable while others may not be.			



OE_TC_084				
Steps	Expected Results	Actual Results	Comments	Test Result
5. If the device does not support multiple allocations of any device capacity, then verify that the <i>deallocateCapacity()</i> operation returns the selected capacity to its respective full capacity.	<p><b>Pass:</b> The <i>deallocateCapacity()</i> operation restore specified capacities. (OE0411)</p> <p><b>Fail:</b> The <i>deallocateCapacity()</i> operation fails to restore specified capacities. (OE0411)</p>		This is descriptive of the simplest form of device, every allocated capacity is either <allocated/ not available> or <not allocated/ available>.	
6. If the device supports multiple allocations of any capacity, then verify that the <i>allocateCapacity()</i> and <i>deallocateCapacity()</i> keeps track how much of said capacity is allocated and deallocated in each operational call.	<p><b>Pass:</b> The <i>deallocateCapacity()</i> operation restores specified capacities by their prescribed quantities and to not more than their full quantity. (OE0411)</p> <p><b>Fail:</b> The <i>deallocateCapacity()</i> operation fails to restore specified capacities by their prescribed quantities. (OE0411)</p> <p><b>Fail:</b> The <i>deallocateCapacity()</i> operation allows specified capacities to be restored in excess of their original capacity. (OE0411)</p>		Accurate accounting must consider further complications; allocations and deallocations must be coordinated; an unpaired deallocation operation must not allow a back door to creating capacity for a future allocate operation	
<b>End of Test</b>				

Test Recording Log – OE_TC_084				
Step1 Device	Step2 Source Code File(s)	Step4 Shared Device Possible?	Step5 Capacities Deallocated when multiple allocations are not supported?	Step6 Capacities Deallocated when multiple allocations are supported?
		Y / N		
		Y / N		
		Y / N		
		Y / N		
		Y / N		
		Y / N		
		Y / N		
		Y / N		

## Test Summary – OE\_TC\_084

Once testing is complete for every component of the OE under test, report the test result as follows:

Pass: No failures detected  
Fail: Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
Untested: Condition which is not testable  
N/A: Not Applicable

**Overall Test Result** (Pass, Fail, Untested, or N/A): \_\_\_\_\_

OE0411 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.17. OE\_TC\_085 - Device :: deallocateCapacity usageState****Test Case Number:** OE\_TC\_085

Device::deallocateCapacity

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0412	The <i>deallocateCapacity</i> operation shall set the <i>usageState</i> attribute to ACTIVE when, after adjusting capacities, any of the device's capacities are still being used.
OE0413	The <i>deallocateCapacity</i> operation shall set the <i>usageState</i> attribute to IDLE when, after adjusting capacities, none of the device's capacities are still being used.
OE0414	The <i>deallocateCapacity</i> operation shall set the <i>adminState</i> attribute to LOCKED as specified in 3.1.3.3.4.4.2.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2, 15 May 2006	Page 89, Section 3.1.3.3.1.5.2.3

**Test Objective**

This test case verifies requirements OE0412, OE0413, and OE0414. The objective is to test that the *deallocateCapacity* operation properly sets the *usageState* and *adminState*. The *usageState* definition allows for a device to be Active, but still have additional capacity available – it is conceivable that an SCA-compliant device could be constructed for which all of its component capacities are allocated in their entirety, such a device could be BUSY or IDLE, but never ACTIVE. The *adminState* indicates the permission to use or prohibition against using the device; it can be set to LOCKED and UNLOCKED values, and transiently holds the value of SHUTTING\_DOWN.

NOTE: The section number in the text of OE0414 is in error. The proper section number is 3.1.3.3.1.4.2.

**Places to Verify**

Device

**IDL References****Data**

struct DataType {

```
    string id;  
    any value; };  
typedef sequence <DataType> Properties;
```

### Exceptions

```
exception InvalidCapacity { string msg; Properties capacities; };  
exception InvalidState { string msg; };
```

### Operations

```
void deallocateCapacity ( in Properties capacities ) raises (InvalidCapacity, InvalidState);
```

### Preconditions

- The source code files are available.

### Test Description

- A. Identify the Devices that implement *allocateCapacity* and *deallocateCapacity* operations. (OE0412, OE0413, OE0414)
- B. Determine if the *allocateCapacity* and *deallocateCapacity* provide support for multiple allocations of any capacities of the device (OE0412).
  1. **N/A:** If the device does not support multiple allocations of a capacity, then the ACTIVE state is never going to be the result of a *deallocateCapacity* operation; a single allocation will result in the BUSY state.
  2. Starting from a device state of ACTIVE or BUSY, verify that the *deallocateCapacity* will set the state to ACTIVE if the capacity is still in use.
    - a. **Pass:** The *deallocateCapacity* operation sets the usageState attribute to ACTIVE when, after adjusting capacities, any of the device's capacities are still being used.
    - b. **Fail:** The *deallocateCapacity* operation sets the usageState attribute to any value other than ACTIVE when, after adjusting capacities, any of the device's capacities are still being used.
- C. Verify that the usageState gets set to IDLE. (OE0413).
  1. Starting from a device state of ACTIVE or BUSY, verify that the *deallocateCapacity* operation sets the usageState attribute to IDLE when, after adjusting capacities, none of the device's capacities are still being used.
    - a. **Pass:** The *deallocateCapacity* operation sets the usageState attribute to IDLE when, after adjusting capacities, none of the device's capacities are still being used.
    - b. **Fail:** The *deallocateCapacity* operation sets the usageState attribute to any value other than IDLE when, after adjusting capacities, none of the device's capacities are still being used.

D. Verify that the adminState gets set to LOCKED. (OE0414).

1. Starting from a device state of ACTIVE or BUSY, verify that the *deallocateCapacity* operation sets the adminState attribute to LOCKED only when the adminState has previously been set to SHUTTING\_DOWN (awaiting the usageState to transition from ACTIVE or BUSY to IDLE) and all child devices have also been set to locked.
  - a. **Pass:** The *deallocateCapacity* operation sets the adminState attribute to LOCKED.
  - b. **Fail:** The *deallocateCapacity* operation does not set the adminState attribute to LOCKED.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_085				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the Devices that implement <i>allocateCapacity</i> and <i>deallocateCapacity</i> operations. (OE0412, OE0413, OE0414)</b>				
<b>B. Determine if the <i>allocateCapacity</i> and <i>deallocateCapacity</i> provide for multiple uses of the device (OE0412).</b>				
1. Determine if the device supports multiple uses, if it does not, then an ACTIVE state does not necessarily have to exist.	<b>Untested:</b> If the device does not support multiple allocations of a capacity, then the ACTIVE state is never going to be the result of a <i>deallocateCapacity</i> operation. (OE0412)			
2. Verify that the <i>deallocateCapacity</i> operation, when starting from a device state of ACTIVE or BUSY, will set the state to ACTIVE if the capacity is still in use.	<b>Pass:</b> The <i>deallocateCapacity</i> operation sets the usageState attribute to ACTIVE when, after adjusting capacities, any of the device's capacities are still being used. (OE0412)  <b>Fail:</b> The <i>deallocateCapacity</i> operation sets the usageState attribute to any value other than ACTIVE when, after adjusting capacities, any of the device's capacities are still being used. (OE0412)			
<b>C. Verify that the usageState gets set to IDLE (OE0413).</b>				

OE_TC_085				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify that the <i>deallocateCapacity</i> operation, when starting from a device state of ACTIVE or BUSY, sets the <i>usageState</i> attribute to IDLE when, after adjusting capacities, none of the device's capacities are still being used.	<p><b>Pass:</b> The <i>deallocateCapacity</i> operation sets the <i>usageState</i> attribute to IDLE when, after adjusting capacities, none of the device's capacities are still being used. (OE0413)</p> <p><b>Fail:</b> The <i>deallocateCapacity</i> operation sets the <i>usageState</i> attribute to any value other than IDLE when, after adjusting capacities, none of the device's capacities are still being used. (OE0413)</p>			
<b>D. Verify that the <i>adminState</i> gets set to LOCKED (OE0414).</b>				
4. Verify that the <i>deallocateCapacity</i> operation, when starting from a device state of ACTIVE or BUSY, sets the <i>adminState</i> attribute to LOCKED only when the <i>adminState</i> has previously been set to SHUTTING_DOWN (awaiting the <i>usageState</i> to transition from ACTIVE or BUSY to IDLE) and all child devices have also been set to locked.	<p><b>Pass:</b> The <i>deallocateCapacity</i> operation sets the <i>adminState</i> attribute to LOCKED. (OE0414)</p> <p><b>Fail:</b> The <i>deallocateCapacity</i> operation does not set the <i>adminState</i> attribute to LOCKED. (OE0414)</p>			
<b>End of Test</b>				



Test Recording Log – OE_TC_085					
Device	Step 1 Multiple Allocations (Y/N)	Step 1 Active State (Y/N)	Step 2 Sets to ACTIVE (Y/N)	Step 3 Sets to IDLE (Y/N)	Step 4 Sets to LOCKED (Y/N)

**Test Summary OE\_TC\_085**

Once testing is complete for every component of the OE under test, report the test result as follows:

Pass: No failures detected  
Fail: Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
Untested: Condition which is not testable  
N/A: Not Applicable

**Overall Test Result** (Pass, Fail, Untested, or N/A): \_\_\_\_\_

OE0412 \_\_\_\_\_

OE0413 \_\_\_\_\_

OE0414 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.18. OE\_TC\_086 - Device :: deallocateCapacity raises InvalidCapacity****Test Case Number:** OE\_TC\_086**CF::** Device::deallocateCapacity raises InvalidCapacity**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0415	The <i>deallocateCapacity</i> operation shall raise the InvalidCapacity exception, when the capacity ID is invalid or the capacity value is the wrong type.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2, 15 May 2006	Page 3-64, Section 3.1.3.3.1.5.2.5 (primary) Page 3-63 through 3-64, Section 3.1.3.3.1.5.2
SCA Appendix C: Core Framework IDL	FINAL/ 15 May 2006, V2.2.2	Pages C-25, C-27

**Test Objective**

This test case verifies OE0415. The objective of this test is to verify that the *deallocateCapacity* operation will raise the InvalidCapacity exception, when the capacity ID is invalid or the capacity value is the wrong type.

**Places to Verify**

Device

**IDL References****Exceptions**

```
exception InvalidCapacity {  
    string msg;  
    CF::Properties capacities;  
};
```

**Operations**

```
void deallocateCapacity (  
    in CF::Properties capacities
```

)  
raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);

## Preconditions

- The OE source code files are available.

## Test Description

A. Determine the devices that the OE provides. (OE0415)

1. **N/A:** No devices are provided.

Note: For each device found, perform the following steps.

B. Verify that the source code for the *deallocateCapacity* operation and *InvalidCapacity* exception is found. (OE0415)

1. **Pass:** The source code is found.
2. **Failed:** The source code is not found.

C. Verify that if a capacity ID is invalid, the *InvalidCapacity* exception will be raised (OE0415).

1. **Pass:** The *InvalidCapacity* exception is raised for an invalid capacity ID.
2. **Fail:** The *InvalidCapacity* exception is not raised for an invalid capacity ID.

D. Verify that if a capacity ID is valid, but the capacity value is of an invalid type, the *InvalidCapacity* exception will be raised (OE0415).

1. **Pass:** The *InvalidCapacity* exception is raised for an invalid capacity ID type.
2. **Fail:** The *InvalidCapacity* exception is not raised for an invalid capacity ID type.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_086				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine the devices that the OE provides. (OE0415)</b>				
1. Examine the DCD file(s) to determine the devices provided by the OE and record the SPD file associated with each device.	<b>Pass:</b> Devices are provided. (OE0415)  <b>N/A:</b> No devices are provided. (OE0415)		Note that a single DCD file may include references to multiple devices.	
2. Find the code element in each SPD and record the local filename.	<b>Fail:</b> No code element is present. (OE0415)		The code element is required in the SPD, so its absence means a failure. If the domain profile tests have passed, then this should never occur.	
<b>Note: For each device found, perform the following steps:</b>				
<b>B. Verify that the source code for the <i>deallocateCapacity</i> operation and <i>InvalidCapacity</i> exception is found.</b>				
3. Locate and record the location of the source code file and directory of the <i>deallocateCapacity</i> operation.	<b>Pass:</b> File is found. (OE0415)  <b>Fail:</b> File is not found. (OE0415)		The software developer can assist in locating the appropriate source code. Perhaps the SPD file from the previous step can be used.	
4. Verify that the <i>deallocateCapacity</i> operation is defined to raise the <i>InvalidCapacity</i> exception.	<b>Pass:</b> The <i>deallocateCapacity</i> operation raises the <i>InvalidCapacity</i> exception. (OE0415)  <b>Fail:</b> The <i>deallocateCapacity</i> operation does not raise the <i>InvalidCapacity</i> exception. (OE0415)			
<b>C. Verify that if a capacity ID is invalid, the <i>InvalidCapacity</i> exception will be raised. (OE0415)</b>				

OE_TC_086				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify in the code that if the resource's capacity ID is invalid, then the <i>InvalidCapacity</i> exception will be raised. (OE0415)	<b>Pass:</b> An invalid capacity ID raises the <i>InvalidCapacity</i> exception. (OE0415)  <b>Fail:</b> An invalid capacity ID does not raise the <i>InvalidCapacity</i> exception. (OE0415)			
<b>D. Verify that if a capacity ID is valid, but the capacity value is of an invalid type, the <i>InvalidCapacity</i> exception will be raised (OE0415).</b>				
6. Verify that if a capacity ID is valid, but the associated value is of the wrong type, then the <i>InvalidCapacity</i> exception will be raised. (OE0415)	<b>Pass:</b> A valid capacity ID with the capacity value of the wrong type raises the <i>InvalidCapacity</i> exception. (OE0415)  <b>Fail:</b> A valid capacity ID with the capacity value of the wrong type does not raise the <i>InvalidCapacity</i> exception. (OE0415)			
<b>End of Test</b>				

Test Recording Log – OE_TC_086					
DCD file(s):					
Step 2 (SPD code element)	Step 3 ( <i>deallocateCapacity</i> )	Step 4 ( <i>InvalidCapacity</i> )	Step 5 (raise exception, in valid ID)	Step 6 (raise exception, in valid type)	Notes

## Test Summary OE\_TC\_086

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of its requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0415\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_



### B.3.19. OE\_TC\_087 - Device :: releaseObject

**Test Case Number:** OE\_TC\_087

**CF:: Device::releaseObject**

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0420	The <i>releaseObject</i> operation shall cause the device to be unavailable and released from the CORBA environment when the <i>Device</i> adminState attribute transitions to LOCKED.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-64 through 3-67, Section 3.1.3.3.1.5.3.3
SCA Appendix C: Core Framework IDL	FINAL/ 15 May 2006, V 2.2.2	C-14
SCA Appendix D: Domain Profile	FINAL/ 15 May 2006, V 2.2.2	

#### Test Objective

This test case verifies OE0420. The objective of this test is to verify that the *releaseObject* operation will cause the Device to be unavailable when the Device's adminState transitions to LOCKED.

#### Places to Verify

Devices

#### IDL References

##### Operations

void *releaseObject* ()

raises (CF::LifeCycle::ReleaseError);

#### Preconditions

- The OE source code and XML files are available.

## Test Description

- A. Determine the number of, names and location of the devices of the Operating Environment from the DCD file(s) of the Domain Profile and/or developer documentation. (OE0420)
1. **Pass:** The DCD file(s) exist and indicates the names of the devices
  2. **Fail:** If there is a DCD file and there are no listed devices.
  3. **Fail:** If there is a listed device, but there are no DCD file(s).

For each device:

- B. Verify that the source code containing the *releaseObject* operation can be found. (OE0420)
1. **Pass:** The source code containing the *releaseObject* operation is found
  2. **Fail:** The source code does not contain the *releaseObject* operation.
- C. Verify that, within the *releaseObject* operation, the Device's *adminState* transition from a state that is not a 'LOCKED' state to a 'LOCKED' state causes the Device to be unavailable and released from the CORBA environment. (OE0420).
1. **Pass:** The transition to a LOCKED state triggers the Device to become unavailable and released from the CORBA environment.
  2. **Fail:** The transition to a LOCKED state does not trigger the Device to become unavailable and released from the CORBA environment.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_087				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine the number of, names and location of the devices of the Operating Environment from the DCD file(s) of the Domain Profile and/or developer documentation. (OE0420)</b>				
1. Locate the DCD file(s) and record the name of each device.	<p><b>Pass:</b> The identified devices in each Device Manager's DCD file are located. In addition, each device listed in the Device Manager's DCD files is located in the source code. (OE0420)</p> <p><b>Fail:</b> No DCD file/source code match-ups are found. (OE0420)</p>		<p>Check all device managers for all of their associated Device Configuration Description (DCD).</p> <p>A device's id and name are usually specified in the DCD file as in the following example:</p> <pre>&lt;componentplacement&gt;   &lt;componentfileref refid="DCE:ddddxxxx"/&gt;   &lt;componentinstantiation id="DCE.ddd123xxxx"&gt;     &lt;usagename&gt;DeviceName&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt; &lt;componentfiles&gt; &lt;componentfile id="DCE:ddddxxxx " type="SPD"&gt;   &lt;localfile name=".../logservice/DeviceFile.spd.xml"/&gt; &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre> <p>Note: Using the name of the component placementref and information contained in the SPD file could also give additional information on how to locate the source code.</p>	

OE_TC_087				
Steps	Expected Results	Actual Results	Comments	Test Result
2. Locate the source code for the device and record the name of the file(s).	<b>Pass:</b> The source code of the devices is located. (OE0420)  <b>Untested:</b> The source code of the devices is not located. (OE0420)		The development engineer and/or the developer documentation, if available, is/are the best source for this information.	
<b>For each device:</b>				
<b>B. Verify that the source code containing the <i>releaseObject</i> operation can be found. (OE0420)</b>				
3. Locate the Device-related <i>releaseObject</i> operation source code.	<b>Pass:</b> The source code containing the <i>releaseObject</i> operation is found. (OE0420)  <b>Fail:</b> If the source code is NOT found, then the test should fail since the <i>releaseObject</i> operation is required for all devices. (OE0420)		For OE, <i>Resource::releaseObject</i> and <i>Lifecycle::releaseObject</i> contain the operation behaviors and should be in the source code, but this test case refers to the <i>Device::releaseObject</i> code.  Device inherits from Resource, which inherits from Lifecycle. As such, the Device interface has additional capabilities and attributes that are device-specific than just LifeCycle-related. So, a <i>releaseObject</i> must also be found within the Device-related source code.	
<b>C. Verify that, within the <i>releaseObject</i> operation, the Device's adminState transition from a state that is not a 'LOCKED' state to a 'LOCKED' state causes the Device to be unavailable and released from the CORBA environment. (OE0420).</b>				
4. Verify that there is logic within the <i>releaseObject</i> operation to check whether the device's adminState is LOCKED.	<b>Pass:</b> The LOCKED adminState logic is found. (OE0420)  <b>Fail:</b> The LOCKED adminState logic is not found. (OE0420)		Values of adminStates are: LOCKED, UNLOCKED and SHUTTING_DOWN state.	

OE_TC_087				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify for the Device that when the adminState attribute transitions to LOCKED, the Device becomes unavailable.	<p><b>Pass:</b> The transition to a LOCKED state triggers the Device to become unavailable. (OE0420)</p> <p><b>Fail:</b> The transition to a LOCKED state does not trigger the Device to become unavailable. (OE0420)</p>		<p>Definition of 'Unavailable': "The Device is released from the CORBA environment, and the logical Device's process is terminated on the OS when applicable."</p> <p>Some clues for being 'unavailable' is that there is a Device state change (stateChangeFrom to stateChangeTo) of the Shutting down status of the adminState attribute.</p>	
6. Verify that the device is released from the CORBA environment.	<p><b>Pass:</b> The device is released. (OE0420)</p> <p><b>Fail:</b> The device is not released. (OE0420)</p>		<p>Some things to check for to verify this release from the CORBA environment:</p> <ul style="list-style-type: none"> <li>• Objects deleted</li> <li>• Destructors called</li> <li>• For each new, buffer created for the device is deleted</li> <li>• var go out of scope and deleted</li> </ul>	
<b>End of Test</b>				

Test Recording Log – OE_TC_087						
Step 1 (device name(s))	Step 2 (name of files)	Step 3 ( <i>releaseObject</i> )	Step 4 (‘LOCKED’ state change)	Step 5 (LOCKED, unavailable Device)	Step 6 (CORBA release)	Notes

## Test Summary OE\_TC\_087

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0420 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.20. OE\_TC\_088 - Device :: adminState attribute changes

**Test Case Number:** OE\_TC\_088

Device::adminState

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0390	The device shall send a StateChangeEvent Type event to the Incoming Domain Management event channel, whenever the adminState attribute changes.
OE0390-C087	The <i>producerId</i> field is the identifier attribute of the device.
OE0390-C088	The <i>sourceId</i> field is the identifier attribute of the device.
OE0390-C089	The <i>stateChangeCategory</i> field is “ADMINISTRATIVE STATE EVENT”.
OE0390-C090	The <i>stateChangeFrom</i> field is the value of the adminState attribute before the state change
OE0390-C091	The <i>stateChangeTo</i> field is the value of the adminState attribute after the state change.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-60, Section 3.1.3.3.1.4.2; Page 3-61, Figure 3-21
SCA Appendix D: Domain Profile	FINAL/ 15 May 2006, V2.2.2	

#### Test Objective

This test case verifies OE0390 and its criteria, OE0390-C087, OE0390-C088, OE0390-C089, OE0390-C090, and OE0390-C091. The objective of this test is to verify that a state change event is sent to the Incoming Domain Management channel (IDM\_CHANNEL) when a device's state changes and that all the required fields in the state change event message are set correctly.

#### Places to Verify

All devices

#### IDL Interface

##### Data

```
struct StateChangeEvent Type {  
    string producerId;
```



```
string sourceId;
StandardEvent::StateChangeCategoryType stateChangeCategory;
StandardEvent::StateChangeType stateChangeFrom;
StandardEvent::StateChangeType stateChangeTo;
};
attribute CF::Device::AdminType adminState;
enum AdminType {
    LOCKED,
    SHUTTING_DOWN,
    UNLOCKED
};
```

## Preconditions

- The DCD and SPD XML files are available.
- The source code files for all devices are available.

## Test Description

- A. Determine the devices provided by the OE and identify the source code that modifies the adminState. (OE0390)
1. **Pass:** Devices are provided and the source code can be identified.
  2. **Fail:** No source code to modify the adminState is provided.
  3. **N/A:** No devices are provided.

Note: For each device found, perform the following steps:

- B. Verify that when an adminState attribute change occurs, that a StateChangeEvent event is sent. (OE0390)
- a. **Pass:** A StateChangeEvent event is sent.
  - b. **Fail:** A StateChangeEvent event is not sent.
- C. Verify that the *producerId* field is the identifier attribute of the device. (OE0390-C087)
- a. **Pass:** The *producerId* field is the identifier attribute of the device.
  - b. **Fail:** The *producerId* field is not the identifier attribute of the device.
- D. Verify that the *sourceId* field is the identifier attribute of the device. (OE0390-C088)
- a. **Pass:** The *sourceId* field is the identifier attribute of the device.
  - b. **Fail:** The *sourceId* field is not the identifier attribute of the device.
- E. Verify that the *stateChangeCategory* field is “ADMINISTRATIVE\_STATE\_EVENT”. (OE0390-C089)
- a. **Pass:** The *stateChangeCategory* field is “ADMINISTRATIVE\_STATE\_EVENT”.

- b. **Fail:** The *stateChangeCategory* field is not “ADMINISTRATIVE\_STATE\_EVENT”.
- F. Verify that the *stateChangeFrom* field is the value of the adminState attribute before the state change. (OE0390-C090)
  - a. **Pass:** The *stateChangeFrom* field is the value of the adminState attribute before the state change.
  - b. **Fail:** The *stateChangeFrom* field is not the value of the adminState attribute before the state change.
- G. Verify that the *stateChangeTo* field is value of the adminState attribute after the state change. (OE0390-C091)
  - a. **Pass:** The *stateChangeTo* field is the value of the adminState attribute after the state change.
  - b. **Fail:** The *stateChangeTo* field is not the value of the adminState attribute after the state change.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_088				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine the devices provided by the OE and identify the code that modifies the adminState. (OE0390)</b>				
1. Examine the DCD files to determine the devices provided by the OE and record the SPD file associated with each device.	<b>Pass:</b> Devices are provided. (OE0390)  <b>Untested:</b> No devices are provided. (OE0390)		Note that a single DCD file may include references to multiple devices.	
2. Find the code element in the SPD and record the local filename for each device.	<b>Fail:</b> No code element is present. (OE0390)		The code element is required in the SPD, so its absence means a failure.	
3. Locate the source code that modifies the adminState and record the source file name for each device.	<b>Pass:</b> The source code can be located. (OE0390)  <b>Fail:</b> No source code for modifying the adminState can be located. (OE0390)		In the absence of an OE software engineer, the local filename recorded in step 2 is a clue to where the source code is located.	
<b>Note: For each device found, perform the following steps:</b>				
<b>B. Verify that when an adminState attribute change occurs, that a StateChangeEvent event is sent. (OE0390)</b>				
4. Verify that, when the adminState is modified, a StateChangeEvent event is sent to the IDM_channel.	<b>Pass:</b> A StateChangeEvent event is sent. (OE0390)  <b>Fail:</b> A StateChangeEvent event is not sent. (OE0390)			
<b>C. Verify that the producerId field is the identifier attribute of the device (OE0390-C087).</b>				

OE_TC_088				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify that the <i>producerId</i> field is the identifier attribute of the device.	<b>Pass:</b> The <i>producerId</i> field is the identifier attribute of the device. (OE0390-C087)  <b>Fail:</b> The <i>producerId</i> field is not the identifier attribute of the device. (OE0390-C087)		Failure of criterion OE0390-C087 means failure of the requirement OE0390.	
<b>D. Verify that the <i>sourceId</i> field is the identifier attribute of the device (OE0390-C088).</b>				
6. Verify that the <i>sourceId</i> field is the identifier attribute of the device.	<b>Pass:</b> The <i>sourceId</i> field is the identifier attribute of the device. (OE0390-C088)  <b>Fail:</b> The <i>sourceId</i> field is not the identifier attribute of the device. (OE0390-C088)		Failure of criterion OE0390-C088 means failure of the requirement OE0390.	
<b>E. Verify that the <i>stateChangeCategory</i> field is “ADMINISTRATIVE_STATE_EVENT” (OE0390-C089).</b>				
7. Verify that the <i>stateChangeCategory</i> field is “ADMINISTRATIVE_STATE_EVENT”.	<b>Pass:</b> The <i>stateChangeCategory</i> field is “ADMINISTRATIVE_STATE_EVENT”. (OE0390-C089)  <b>Fail:</b> The <i>stateChangeCategory</i> field is not “ADMINISTRATIVE_STATE_EVENT”. (OE0390-C089)		Failure of criterion OE0390-C089 means failure of the requirement OE0390.	
<b>F. Verify that the <i>stateChangeFrom</i> field is the value of the <i>adminState</i> attribute before the state change (OE0390-C090).</b>				

OE_TC_088				
Steps	Expected Results	Actual Results	Comments	Test Result
8. Verify that the <i>stateChangeFrom</i> field is the value of the <i>adminState</i> attribute before the state change.	<p><b>Pass:</b> The <i>stateChangeFrom</i> field is the value of the <i>adminState</i> attribute before the state change. (OE0390-C090)</p> <p><b>Fail:</b> The <i>stateChangeFrom</i> field is not the value of the <i>adminState</i> attribute before the state change. (OE0390-C090)</p>		Failure of criterion OE0390-C090 means failure of the requirement OE0390.	
<b>G. Verify that the <i>stateChangeTo</i> field is the identifier attribute of the device (OE0390-C091).</b>				
9. Verify that the <i>stateChangeTo</i> field is the identifier attribute of the device.	<p><b>Pass:</b> The <i>stateChangeTo</i> field is the value of the <i>adminState</i> attribute after the state change. (OE0390-C091)</p> <p><b>Fail:</b> The <i>stateChangeTo</i> field is not the value of the <i>adminState</i> attribute after the state change. (OE0390-C091)</p>		Failure of criterion OE0390-C091 means failure of the requirement OE0390.	
<b>End of Test</b>				

Test Recording Log – OE_TC_088								
Step 1 (SPD File name)	Step 2 (Code element name)	Step 3 (source file name)	Step 4 (event sent)	Step 5 (producerId)	Step 6 (sourceId)	Step 7 (state Change Category)	Step 8 (state Change From)	Step 9 (state Change To)

## Test Summary OE\_TC\_088

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0390 \_\_\_\_\_

OE0390-C087 \_\_\_\_\_

OE0390-C088 \_\_\_\_\_

OE0390-C089 \_\_\_\_\_

OE0390-C090 \_\_\_\_\_

OE0390-C091 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.21. OE\_TC\_089 - Device :: releaseObject raises ReleaseError exception****Test Case Number:** OE\_TC\_089

Device::releaseObject raises ReleaseError exception

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0423	The <i>releaseObject</i> operation shall raise the <i>ReleaseError</i> exception when <i>releaseObject</i> is not successful in releasing a logical device due to internal processing errors that occurred within the device being released.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-10, Section 3.1.3.1.2.5.2; Page 3-63, Section 3.1.3.3.1.5.3;

**Test Objective**

This test case verifies OE0423. The objective of this test is to verify that a *releaseObject* will raise the *ReleaseError* exception when an internal processing error prevents successful release of a logical device.

**Places to Verify**

Logical devices

**IDL References****Exceptions**

exception ReleaseError { CF::StringSequence errorMessages; };

**Operations**

void releaseObject () raises (CF::LifeCycle::ReleaseError);

**Preconditions**

- The source code files are available.



## Test Description

A. Determine the devices that the OE provides. (OE0423)

1. **N/A:** No devices are provided.

Note: For each device found, perform the following steps.

B. Locate the source code for *the releaseObject()* operation. (OE0423)

1. **Pass:** source code can be located.
2. **Fail:** No source code is located.

C. Verify that the *ReleaseError* exception is raised if the logical device is not released. (OE0423)

1. **Pass:** The *ReleaseError* exception is raised when the logical device is not released.
2. **Fail:** The *ReleaseError* exception is not raised when the logical device is not released.
3. **Fail:** The *ReleaseError* exception is raised when the logical device is released.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_089				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine the devices that the OE provides. (OE0423)</b>				
1. Examine the DCD files to determine the devices provided by the OE and record the SPD file associated with each device.	<b>Pass:</b> Devices are provided. (OE0423)  <b>Untested:</b> No devices are provided. (OE0423)		Note that a single DCD file may include references to multiple devices.	
2. Find the code element in each SPD and record the local filename.	<b>Fail:</b> No code element is present. (OE0423)		The code element is required in the SPD, so its absence means a failure. If the domain profile tests have passed, then this should never occur.	
<b>Note: For each device found, perform the following steps:</b>				
<b>B. Locate the source code for the <code>releaseObject()</code> operation. (OE0423)</b>				
3. Locate the source code for <i>releaseObject</i> and record the source file name.	<b>Pass:</b> The source code can be located. (OE0423)  <b>Fail:</b> No source code for <i>releaseObject</i> can be located. (OE0423)		In the absence of an OE software engineer, the local filename recorded in step 2 is a clue to where the source code is located.	
<b>C. Verify that the <code>ReleaseError</code> is raised if the logical device is not released. (OE0423)</b>				

OE_TC_089				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Verify that if the logical device is not released that a <i>ReleaseError</i> is raised.	<b>Pass:</b> The <i>ReleaseError</i> is raised properly. (OE0423)  <b>Fail:</b> The <i>ReleaseError</i> is not raised when the logical device is not released. (OE0423) <b>Fail:</b> The <i>ReleaseError</i> is raised when the object is released. (OE0423)			
<b>End of Test</b>				

Test Recording Log – OE_TC_089				
Step 1 (SPD file)	Step 2 (Code element)	Step 3 ( <i>releaseObject</i> found)	Step 4 ( <i>ReleaseError</i> )	Notes

## Test Summary OE\_TC\_089

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0423 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.22. OE\_TC\_092 - Device :: Logical Device executable parameters

**Test Case Number:** OE\_TC\_092

Device::Logical Device

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0618	The executable parameters of a logical device shall accept the standard argv arguments as used in the POSIX exec family of functions [4].
OE0619	A logical device shall accept the executable parameters as specified in section 3.1.3.3.5.1.3 (ExecutableDevice::execute).

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-96, Section 3.3; Page 3-97, Section 3.3.1; Page 3-72, Section 3.1.3.3.5.1.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pgs. C-2, C-32, Section C.1
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-9, Section D.2.1.6.3
ISO/IEC 9945-1	2003 Edition	Pages 38, 313

#### Test Objective

This test case verifies OE0618 and OE0619. The objective of this test is to verify that the logical device which are also executable devices accept the standard argv arguments as described by the POSIX exec family of functions (OE0618), and also verifies that the ordering of the argv arguments is in the format that is specified in the SCA v2.2.2, Section 3.1.3.3.5.1.3 (OE0619). Executable devices are logical devices but not all logical devices are executable devices. A logical device inherits from at least one of the following interfaces: *Device*, *LoadableDevice*, *ExecutableDevice*, and *AggregateDevice*. For the purposes of this test procedure, the logical devices under test are also executable devices. The standard argv format accepts an array of character pointers to null terminated strings. The ordering of arguments, according to argv format, should list argv(0) as the function name. The remaining input parameters, which are id/value pairs, should follow the same consecutive ordering, with the id as the next entry and the value of the pair as the entry following the id. The pattern effectively works out to be the ids as the odd numbered entries and the values as the even number entries, excluding the first entry, which is the function/file name.

#### Places to Verify

Executable Devices

## IDL References

### Data

```
struct DataType { string id; any value; };  
typedef sequence <DataType> Properties;
```

## Preconditions

- The Domain Profile files are available
- The source code files of the Operating Environment are available

## Test Description

A. Identify the executable devices of the Operating Environment. (OE0618, OE0619)

1. **N/A:** The OE does not contain executable devices or does not contain devices at all.
2. **Untested:** The source code of the executable devices cannot be located.

For each of the executable devices in the Operating Environment, perform the following steps:

B. Verify that the executable device's entry point arguments, which are id/value string pairs, conform to the standard argv of the POSIX exec family of functions. (OE0618)

**NOTE:** The entry point of an executable device is a function that starts the execution of the device.

1. **Pass:** The executable device's entry point arguments conform to the standard argv of the POSIX exec family of functions.
2. **Fail:** The executable device's entry point arguments do *not* conform to the standard argv of the POSIX exec family of functions.

C. Verify that the executable device's entry point arguments conform to the format specified in section 3.1.3.3.3.5.1.3, which places the first input argument as the file or function name and places the subsequent arguments(id/value string pairs), into an alternating pattern, with the value entry following each id that it corresponds to. (OE0619)

1. Verify that the first input parameter, argv(0), is mapped to the file or function name. (OE0619)
  - a. **Pass:** The first input parameter, argv(0), is mapped to the file or function name.
  - b. **Fail:** The first input parameter, argv(0), is not mapped to the file or function name.
2. Verify that the id of the first id/value pair maps to the next argument (argv(1)), the one following the file or function name, and the value of the first id/value pair maps to the argument following the id(argv(2)). All remaining id/value pairs should be mapped using this alternating pattern until all id/value pairs are used. This effectively works out that the ids are the odd numbered entries and the values are the even numbered entries. (OE0619)
  - a. **Pass:** The first id/value pair is mapped to the list of arguments, with the id as argv(1) and the value as argv(2). All of the following id/value pairs map to this alternating pattern, with ids as odd entries and values as even entries.
  - b. **Fail:** The first id/value pair is not mapped with id as argv(1) and value as argv(2).

- c. **Fail:** Any of the following id/value pairs after the first id/value pairs is not mapped to the alternating pattern.



## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_092				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the executable devices of the Operating Environment. (OE0618, OE0619)</b>				
1. Consult with developer and/or examine the DCD files to identify the devices provided by the OE.	<p>Finding all the devices is a helpful first step in finding the executable devices in the OE.</p> <p><b>N/A:</b> There are no devices in the OE. (OE0618, OE0619)</p>		<p><b>NOTE:</b> The fastest way to determine the device names is to consult with the developer and OE documentation, although it may be necessary to examine the XML files to locate the names and SPD files of the device(s).</p> <p>A device's id and name are usually specified in the DCD file as in the following example. The following data used is sample data, not actual.</p> <pre>&lt;componentplacement&gt;   &lt;componentfileref refid="yyyyy"/&gt;   &lt;componentinstantiation id="DCE:xxxx-xxxx"&gt;     &lt;usagename&gt;DCE:yyyyyyxxxxx &lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt; &lt;componentfiles&gt; &lt;componentfile id="yyyyy" type="SPD"&gt;   &lt;localfile name=".../logservice/xxxx.spd.xml"/&gt; &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre>	
2. Examine the DCD files to determine the SPD file associated with each device.	Finding the SPD files is a helpful step in finding the executable devices in the OE.		<pre>&lt;componentplacement&gt;   &lt;componentfileref refid="yyyyy"/&gt;   &lt;componentinstantiation id="DCE:xxxxyyyyyy"&gt;     &lt;usagename&gt;zzzz&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt; &lt;componentfiles&gt; &lt;componentfile id="yyyyy" type="SPD"&gt;   &lt;localfile name=".../xxxx.spd.xml"/&gt; &lt;/componentfile&gt;</pre>	

OE_TC_092				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Examine the OE documentation to identify the executable devices or identify the executable devices by examining the Domain Profile files (SPD) and determine if the code type element is equal to the string, "Executable". It may be helpful to record the localfile name and the entrypoint name.	N/A: There are no executable devices. The test ends at this point. (OE0618, OE0619)		<p>The following is an example of an SPD file that has a code type of executable.</p> <pre>&lt;implementation aepcompliance="aep_compliant" id="DCE:xxxxxx-yyyy-xxxx-yyyy-xxxxxxxxxx"&gt;   &lt;code type="Executable"&gt;     &lt;localfile name=       &lt;localfile name="EntryPoint.out"/&gt;       &lt;entrypoint&gt;EntryPoint&lt;/entrypoint&gt;     &lt;/code&gt;     &lt;stacksize&gt;xxxx&lt;/stacksize&gt;     &lt;priority&gt;XX&lt;/priority&gt;      &lt;processor name="XXX"/&gt;&lt;/implementation&gt;</pre> <p>Another possible way to determine the executable devices are to locate the SCA defined <i>execute</i> or <i>terminate</i> operations in either the device or the device's inheritance hierarchy.</p>	
4. Locate the source code for the executable device and record the name of the file.	<b>Untested:</b> The source code for the executable devices is not located. The test ends at this point. (OE0618, OE0619)		<p><b>Note:</b> May need to consult with the developer for the location of the executable devices. Clues provided by the XML entries in steps 1-3 may give additional information on how to locate the source code.</p> <pre>&lt;code type="Executable"&gt;   &lt;localfile name=".../bin/aDevice"/&gt;   &lt;entrypoint&gt;/sampleDevice&lt;/entrypoint&gt; &lt;/code&gt;</pre>	
<b>For each of the executable devices in the Operating Environment, perform the following steps:</b>				
<b>B. Verify that the executable device's entry point arguments, which are id/value string pairs, conform to the argv format of the POSIX exec family of functions. (OE0618)</b> <b>NOTE: The entry point of an executable device is a function that starts the execution of the device.</b>				
5. Locate the entry point of the device. List the entry point of the device.	<b>Untested:</b> The entry point of the device is not found. (OE0618)		The entry point of the device is usually, but not always, the main function. It may be the same as the entrypoint name from the XML in from step 3.	

OE_TC_092				
Steps	Expected Results	Actual Results	Comments	Test Result
6. Verify in the source code that the executable device's entry point parameter accepts an argument in the argv format.	<p><b>Pass:</b> The executable device's entry point arguments conform to the standard argv of the POSIX exec family of functions. (OE0618)</p> <p><b>Fail:</b> The executable device's entry point parameter does not accept an argument in the argv format. (OE0618)</p>		The standard definition of the argv format is an array (or equivalent data structure) of character pointers to strings ending with a null pointer.	
<p><b>C. Verify that the executable device's entry point arguments conform to the format specified in section 3.1.3.3.5.1.3, which places the first input argument as the file or function name and places the subsequent arguments(id/value string pairs), into an alternating pattern, with the value entry following each id that it corresponds to. (OE0619)</b></p> <ol style="list-style-type: none"> <li>1. Verify that the first input parameter, argv(0), is mapped to the file or function name. (OE0619)</li> <li>2. Verify that the id of the first id/value pair maps to the next argument(argv(1)), the one following the file or function name, and the value of the first id/value pair maps to the argument following the id(argv(2)). All remaining id/value pairs should be mapped using this alternating pattern until all id/value pairs are used. This effectively works out that the ids are the odd numbered entries and the values are the even numbered entries. (OE0619)</li> </ol>				
7. Verify in the source code of the executable device that the first input argument, argv(0), is accepted as the file or function name.	<p><b>Pass:</b> The first input parameter, argv(0), is the file or function name. At this point, only the step passes, not the entire test. (OE0619)</p> <p><b>Fail:</b> The first input parameter, argv(0), is not the file or function name. (OE0619)</p>		It may be useful to look where the entry point also is called to determine the order of the argv arguments, although the implementation of how the entry point handles the argv is the purpose of the test.	

OE_TC_092				
Steps	Expected Results	Actual Results	Comments	Test Result
8. Verify in the source code of the executable device that the first id/value string pair is mapped to the argv string with the id as argv(1) and value as argv(2).	<p><b>Pass:</b> The first id/value pair is mapped to the argv string with the id as argv(1) and the value as argv(2). At this point, only the step passes, not the entire test. (OE0619)</p> <p><b>Fail:</b> The first id/value pair is not mapped to the argv string with id as argv(1) and value as argv(2). (OE0619)</p>		<p>The following is a description from SCA v2.2.2, page 72, and explains the format of the POSIX input parameters.</p> <p>“The execute operation shall convert the input parameters (id/value string pairs) parameter to the standard argv of the POSIX exec family of functions, where argv(0) is the function name. The execute operation shall map the input parameters parameter to argv starting at index 1 as follows, argv(1) maps to input parameters (0) id and argv(2) maps to input parameters (0) value and so forth. The execute operation passes argv through the operating system “execute” function.”</p>	
9. Verify in the source code of the executable device that the remaining id/value pairs are mapped with the ids of each pair being the odd entries and the values of each pair being even entries.	<p><b>Pass:</b> All subsequent id/value entries are mapped as alternating entries: with ids as the odd placed entries and the values as even placed entries. (OE0619)</p> <p><b>Fail:</b> Any of the following id/value pairs after the first id/value pairs is not mapped to the alternating pattern. (OE0619)</p>			
<b>End of Test</b>				

Test Recording Log – OE_TC_092								
Step1 (List of device executable names)	Step2 (SPD file of each device)	Step3 (List of executable devices)	Step4 (Source code executable device file name)	Step5 (List entry point of executable device)	Step6 (accepts standard argv?)  (Y/N?)	Step7 (argv(0) is file or function name?)  (Y/N?)	Step8 (argv(1) is id and argv(2) is value?)  (Y/N?)	Step9 (rest of argv maps to id/value pairs?)  (Y/N?)

## Test Summary OE\_TC\_092

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0618\_\_\_\_\_

OE0619\_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.23. OE\_TC\_094 - Device :: Logical Device - CORBA****Test Case Number:** OE\_TC\_094

Logical Device CORBA

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0620	Logical devices shall be limited to using CORBA and CORBA services defined in the referenced minimum CORBA specification [5].

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Section 3.3.2 – page 3-97
Minimum CORBA Specification	Version 1.0 15 Aug 2002 (02-08-01)	All
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-1, C-5, C-29, C-30
Naming Service Specification	Version 1.3 October 2005 (formal/04-10-03)	Page 2-1 thru 2-4
Event Service Specification	Version 1.2 October 2005 (formal/04-10-02)	Page 1-3, 1-7, Page 2-1 thru 2-5
Lightweight Log Specification	Version 1.1 February 2005 (formal/05-02-02)	Page 2-2 to 2-3

**Test Objective**

This test case verifies OE0620. This test will verify that a logical device is limited to using CORBA as defined in the minimum CORBA specification and CORBA services as defined in the SCA specification. The CORBA services defined here are naming service, event service, and the optional log service. A logical device is a software component that implements one of the Basic Device Interfaces, namely, Device, LoadableDevice, ExecutableDevice, and AggregateDevice.

**Places to Verify**

Devices and any other software component implementing the Base Device Interfaces

**CORBA References****Operations**

*Limited set of minimum CORBA specification*

```

interface LoadableDevice : Device {
interface ExecutableDevice : LoadableDevice {
interface AggregateDevice {
module CF {interface Device;

```

### ***Limited set of CORBA services***

```

CosNaming::BindingIterator
CosNaming::NamingContext
CosNaming::BindingIterator
interface NamingContextExt: NamingContext {
interface LogProducer : LogStatus {
interface LogConsumer : LogStatus {
interface LogAdministrator : LogStatus {
interface LogStatus {
interface PushConsumer {
interface PushSupplier {
interface PullSupplier {
interface PullConsumer {

```

## **Exceptions**

### ***Limited set of minimum CORBA specification***

```

LoadableDevice
    exception InvalidLoadKind { };
    exception LoadFail { CF::ErrorNumberType errorNumber; string msg; };

```

```

ExecutableDevice
    exception InvalidOptions { CF::Properties invalidOpts; };
    exception InvalidParameters { CF::Properties invalidParms; };

```

```

AggregateDevice, interface device
    exception InvalidObjectReference { string msg; };

```

### ***Limited set of CORBA services***

```

NamingContextExt
    exception NotFound { NotFoundReasonwhy; Namerest_of_name; };

```



```
exception CannotProceed { NamingContext ctx; Name rest_of_name; };  
exception InvalidName { };  
exception AlreadyBound { };  
exception NotEmpty { };  
exception InvalidAddress { };
```

LogAdministrator

```
exception InvalidParam { string details; };
```

PushConsumer, PushSupplier, PullConsumer, PullSupplier

```
exception Disconnected { };
```

## Preconditions

- The Core Framework source code files are available.

## Test Description

- A. Identify the components that implement the minimumCORBA specification and CORBA services. (OE0620)
1. **Untested:** There are no components that implement the minimumCORBA specification and/or CORBA services.
- For each component identified above, perform the following step(s):
- B. Verify that the component implements the limited set of minimumCORBA specification. (OE0620)
- a. **Pass:** Only the minimumCORBA interfaces are implemented by the component.
  - b. **Fail:** ORB Helper functions are used.
  - c. **Fail:** Components implements a method that is outside of the minimumCORBA specification.
- C. Verify that the component implements the limited set of CORBA services. (OE0620)
- a. **Pass:** Only CORBA services are implemented by the component.
  - b. **Fail:** Components do not implement a CORBA service method.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_094				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the components of the Core Framework that implement the minimum CORBA specification and CORBA services. (OE0620)</b>				
1. Identify the source code for the Core Framework. Record the name of the file(s).	Source code file(s) are found.  <b>Untested:</b> There are no components that implement the minimum CORBA specification and/or CORBA services. (OE0620)		Consult the developer or review the developer documentation to locate the source code.	
<b>For each component identified above, perform the following step(s):</b>				
<b>B. Verify that the component implements the limited set of minimum CORBA specification. (OE0620)</b>				
2. Identify the components that implements the limited set of minimum CORBA specification. Record the component(s).	<b>Pass:</b> A component that implements the limited set of minimum CORBA specification is found. (OE0620)  <b>Fail:</b> ORB Helper functions are used. (OE0620)  <b>Fail:</b> Components implements a method that is outside of the minimum CORBA specification. (OE0620)		<b>minimum CORBA</b> interface LoadableDevice : Device { interface ExecutableDevice : LoadableDevice { interface AggregateDevice { module CF { interface Device;  <b>Examples:</b> class CF__ExecutableDevice_impl class CF__LoadableDevice_impl	
<b>C. Verify that the component implements the limited set of CORBA services. (OE0620)</b>				

OE_TC_094				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Identify the components that implements the limited set of CORBA services. Record the component(s).	<p><b>Pass:</b> Only CORBA services are implemented by the component. (OE0620)</p> <p><b>Fail:</b> Components do not implement a CORBA service method. (OE0620)</p>		<p>Search for these <b>CORBA services</b>:</p> <p>CosNaming::BindingIterator  CosNaming::NamingContext  CosNaming::BindingIterator  interface NamingContextExt: NamingContext {  interface LogProducer: LogStatus {  interface LogConsumer: LogStatus {  interface LogAdministrator: LogStatus {  interface LogStatus {  interface PushConsumer {  interface PushSupplier {  interface PullSupplier {  interface PullConsumer {</p> <p><b>Example:</b>  CORBA::Object_var obj = orb-&gt;resolve_initial_references("NameService");  _domainFinder._namingService = CosNaming::NamingContext::_narrow(obj);</p>	
End of Test				

Test Recording Log – OE_TC_094		
Step1 (source code files)	Step2 (components implementing minimum CORBA)	Step3 (components implementing CORBA services)

## Test Summary OE\_TC\_094

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0620 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.24. OE\_TC\_095 - Device :: Logical Device - CORBA Register****Test Case Number:** OE\_TC\_095

Logical Device CORBA

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0621	A logical device shall register itself with a device manager using the value associated with the <i>DEVICE_MGR_IOR</i> parameter per 3.1.3.2.4.5.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-98, Section 3.3.3 Page 3-50 thru 53, Section 3.1.3.2.4.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-33, Section C.1
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Pages D-48 thru 49, Section D.7

**Test Objective**

This test case verifies OE0621. The test will verify that a logical device will register itself with a device manager using the parameter *DEVICE\_MGR\_IOR*. A logical device is a software component that implements one of the Basic Device Interfaces, namely, Device, LoadableDevice, ExecutableDevice, and AggregateDevice

**Places to Verify**

Logical Devices

**IDL References****Operations**

void registerDevice (in Device registeringDevice) raises (InvalidObjectReference);

**Exceptions**

CF InvalidObjectReference

## Preconditions

- Domain Profile test is passed
- All the Domain Profile files are available
- The OE source code files having the *registerDevice* operation are available.

## Test Description

- A. Locate all the Device Manager's Device Configuration Descriptor (DCD) files of the OE. (OE0621)
1. **Pass:** One or more DCD file exists.
  2. **Untested:** A DCD file is not provided.

- B. Examine the DCD file and locate all Software Package Descriptor (SPD) files.
1. **Pass:** All the SPD files mentioned in the DCD files are located.
  2. **Fail:** SPD files referenced in the DCD files are not located.

For each device, perform the following:

- C. Examine the SPD file for the source file of the device. (OE0621)
1. **Untested:** Source file for the device is not found.
- D. Locate in the source file for the parameter *DEVICE\_MGR\_IOR* that is used to extract the device's Device Manager. (OE0621)
1. **Pass:** The parameter *DEVICE\_MGR\_IOR* is found.
  2. **Untested:** The parameter *DEVICE\_MGR\_IOR* is not found.
- E. Verify that each device makes a call to the Device Manager's *registerDevice* interface. (OE0621)
1. **Pass:** Each device calls the *registerDevice* interface.
  2. **Fail:** The device does not call the *registerDevice* interface.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_095				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all the Device Manager's Device Configuration Descriptor (DCD) files of the OE.</b>				
1. Locate all the Device Manager's Device Configuration Descriptor (DCD) files of the OE. Record the name of the DCD.	<b>Pass:</b> One or more DCD files exist. (OE0621)  <b>Untested:</b> A DCD file is not provided. (OE0621)		A device's id and name are usually specified in the DCD file as in the following example:  <pre> &lt;componentplacement&gt;   &lt;componentfileref     refid="DCE:ddddxxxx"/&gt;   &lt;componentinstantiation     id="DCE.ddd123xxxx"&gt;     &lt;usagename&gt;<b>DeviceName</b>&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt;           </pre>	
<b>B. Examine each DCD and locate the Software Package Descriptor (SPD) files. (OE0621)</b>				
2. Examine each DCD file and locate all SPD files.	<b>Pass:</b> All the SPD files mentioned in the DCD files are located. (OE0621)  <b>Fail:</b> SPD files referenced in the DCD files are not located. (OE0621)		<pre> &lt;componentfiles&gt;   &lt;componentfile id="DCE:ddddxxxx "     type="SPD"&gt;     &lt;localfile       name=".../xxxxx.spd.xml"/&gt;     &lt;/componentfile&gt; &lt;/componentfiles&gt;           </pre>	
<b>For each device, perform the following:</b>				
<b>C. Examine the SPD file for the source file of the device. (OE0621)</b>				
3. Examine the SPD file for the source file of the device. Record the source file.	<b>Untested:</b> Source file for the device is not found. (OE0621)		The developer may be of assistance in locating the source code.	
<b>D. Identify the source code that launches the device. (OE0621)</b>				



OE_TC_095				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Verify that the <i>DEVICE_MGR_IOR</i> is used to identify the DeviceManager to which the current device belongs.	<b>Pass:</b> The parameter <i>DEVICE_MGR_IOR</i> is found. (OE0621)  <b>Untested:</b> The parameter <i>DEVICE_MGR_IOR</i> is not found. (OE0621)		Look for statements where CORBA::_narrow() is called by using the <i>DEVICE_MGR_IOR</i> as the argument. The returned object should be the device's associated DeviceManager.	
<b>E. Verify that each device makes a call to the DeviceManager's registerDevice interface. (OE0621)</b>				
5. Verify that each device calls <i>registerDevice()</i> .	<b>Pass:</b> The device calls <i>registerDevice()</i> . (OE0621)  <b>Fail:</b> The device does not call <i>registerDevice()</i> (OE0621)		A keyword search of "registerDevice" within the source code file should reveal the statement.	
<b>End of Test</b>				

Test Recording Log – OE_TC_095		
Step1 (DCD files)		
Step3 (source file names)	Step4 ( <i>DEVICE_MGR_IOR</i> used Y/N?)	Step5 (is registerDevice called Y/N?)

Test Recording Log – OE_TC_095		
Step1 (DCD files)		
Step3 (source file names)	Step4 ( <i>DEVICE_MGR_IOR</i> used Y/N?)	Step5 (is registerDevice called Y/N?)

## Test Summary OE\_TC\_095

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0621 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.25. OE\_TC\_096 - Aggregate Device

#### Test Case Number: OE\_TC\_096

Aggregate Device

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0622	A child device shall add itself to a parent device using the executable Composite Device IOR parameter per 3.1.3.2.4.5.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-98, Section 3.3.3; Pages 3-52-3-53, Section 3.1.3.2.4.5; Page 3-75, Section 3.1.3.3.4.5.1.3
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-51; Section D.7.1.4.1.5

#### Test Objective

This test case verifies OE0622. The objective of this test is to verify that a child device will add itself to a parent device using the executable Composite Device IOR parameter per SCA 3.1.3.2.4.5. Section 3.1.3.2.4.5 explains how the Composite\_DEVICE\_IOR is obtained from the entry point of the device in order for the child device to add itself to the aggregate device of the parent.

#### Places to Verify

AggregateDevice\child devices

#### IDL References

##### Data

```
interface AggregateDevice { readonly attribute CF::DeviceSequence devices; }
```

##### Operations

```
void addDevice ( in CF::Device associatedDevice ) raises (CF::InvalidObjectReference);
```

#### Preconditions

- The Domain Profile files are available.

- The source code files of the Core Framework (CF) are available.

## Test Description

**Note:** This test should be evaluated in parallel with the test for OE0474, test case OE\_TC\_034.

A. Identify the devices of the Core Framework. (OE0622)

1. **Untested:** Devices are not present in the Core Framework.

B. Identify the child devices of the Core Framework. (OE0622)

1. **Untested:** The CF does not contain child devices.

For each child device in the Core Framework, perform the following steps:

C. Identify the entry point of the device. (OE0622)

**NOTE:** The entry point of a device is the operation that starts execution of the device.

1. **Untested:** The entry point of the device is not identified.

D. Verify that when the device is launched it obtains the executable Composite\_DEVICE\_IOR parameter according to Section 3.1.3.2.4.5. (OE0622)

1. **Pass:** The device obtains the Composite\_DEVICE\_IOR execute parameter when it is launched.
2. **Fail:** The device is a child device, but does not obtain the Composite\_DEVICE\_IOR execute parameter when it is launched.

E. Verify that the child device adds itself to its parent device using the executable Composite\_DEVICE\_IOR parameter. (OE0622)

1. **Pass:** The child device adds itself to its parent device using the executable Composite\_DEVICE\_IOR parameter.
2. **Fail:** The child device does not add itself to its parent using the Composite\_DEVICE\_IOR parameter.
3. **Fail:** The aggregate device object does not contain the *addDevice* operation.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_096				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the devices of the CF. (OE0622)</b>				
1. Using conversations with the developer, CF documentation and examination of the Domain Profile files, identify the devices of the CF.	<b>Untested:</b> Devices are not present in the OE. (OE0622)		A device's id and name are usually specified in the DCD file as in the following example: <pre>&lt;componentplacement&gt;   &lt;componentfileref refid="CDATA"&gt;     &lt;componentinstantiation id="DCE:dddd-xxxx-yyyy"&gt;       &lt;usagename&gt;aDevice&lt;/usagename&gt;     &lt;/componentinstantiation&gt;   &lt;/componentplacement&gt; &lt;componentfiles&gt;   &lt;componentfile id="&lt;CDATA&gt;" type="SPD"&gt;     &lt;localfile name=" ../DeviceFile.spd.xml"/&gt;   &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre>	
2. Locate the source code for the devices and record the names of the files.	<b>Untested:</b> The source code of the devices is not located. (OE0622)		The development engineer and the developer documentation, if available, is the best source for the location of the source code. In some cases, it is possible to trace the location of the source code from the executable declaration in the SPD.	
<b>B. Identify the child devices. (OE0622)</b>				

OE_TC_096				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Identify the child devices of the CF devices found in steps 1-2 through appropriate means, such as source code search or Domain Profile examination.	<b>Untested:</b> The OE source code does not contain child devices. (OE0622)		<p>One could look in the source code for a CORBA object to be narrowed to the aggregate device interface.</p> <p>A child device relationship could be indicated in the Domain Profile. See reference for SCA v2.2.2, Appendix D, If a componentinstantiation element contains the compositepartofdevice element, it will be a child device. Domain Profile, page D-51: The compositepartofdevice element is used when a parent-child relationship exists between devices to reference the componentinstantiation element that describes the parent device when this device's componentinstantiation element describes the child device."</p>	
<b>For each child device of the Operating Environment, perform the following steps:</b>				
<b>C. Identify the entry point of the device. (OE0622)</b> <b>NOTE: The entry point of a device is the operation that starts execution of the device.</b>				
4. In the source code, identify the existence of the entry point of the device where it is launched. List the operation name that starts the execution of the device.	<b>Untested:</b> The entry point of the device is not located. (OE0622)		<p>The entry point of the device is usually, but not always, the function call main(...)</p> <p>It may be useful to search for the usagename from the comments section of Step 1-2 to find the entry point. It is the device label that the entry point uses for some of its arguments.</p>	
<b>D. Verify that when the device is launched it obtains the executable Composite_DEVICE_IOR parameter according to Section 3.1.3.24.5 (OE0622)</b>				



OE_TC_096				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Examine the source code where the device is launched and verify that the device obtains the executable Composite_DEVICE_IOR parameter(an id/value pair) from the entry point of the device. List the Composite_DEVICE_IOR string/value pair(s).	<p><b>Pass:</b> The device obtains the executable Composite_DEVICE_IOR parameter from the entry point of the device. (OE0622)</p> <p><b>Fail:</b> The device does not obtain the executable Composite_DEVICE_IOR parameter from the entry point of the device. (OE0622)</p>		<p>The device obtains these values from requirement, OE0474.</p> <p>The value of the Composite_DEVICE_IOR string/value pair is the IOR of the aggregate device. This IOR is how the device obtains a reference to the aggregate device of the parent.</p>	
<b>E. Verify that the child device adds itself to its parent device using the executable Composite_DEVICE_IOR parameter. (OE0622)</b>				
6. Verify that the child device uses the Composite_DEVICE_IOR parameter to obtain the object reference of the aggregate device of the parent and adds itself to this reference (therefore adding itself to the parent) using the <i>addDevice</i> operation.	<p><b>Pass:</b> The child device uses the Composite_DEVICE_IOR parameter to obtain the object reference to the aggregate device of the parent and adds itself to this reference using the <i>addDevice</i> operation (OE0622)</p> <p><b>Fail:</b> The child device is not able to use the Composite_Device_IOR parameter to obtain the object reference to the aggregate device. (OE0622)</p> <p><b>Fail:</b> The child device does not use the <i>addDevice</i> operation on the aggregate device to add itself to the parent device. (OE0622)</p> <p><b>Fail:</b> The aggregate device object does not contain the <i>addDevice</i> operation. (OE0622)</p>		<p>Search for strings, such as Composite_DEVICE_IOR or <i>addDevice</i> to find where the child device adds itself to the parent.</p> <p>The parent device contains a reference to the aggregate device. The child device obtains this object reference by resolving the stringified IOR (value of Composite_DEVICE_IOR) in the naming service and then adds itself to this reference.</p>	

---

OE_TC_096				
Steps	Expected Results	Actual Results	Comments	Test Result
End of Test				

Test Recording Log – OE_TC_096						
Step1 (Device name)	Step2 (Source code file of device.)	Step3 (List the child devices, or checkmark if device from step2 is a child)	Step4 (entry point operation name)	Step5 (Device obtains Composite_Devi ce_IOR from entry point?)  (Y/N/?)	Step5 cont'd (List of string value pair)	Step6 (child device obtains reference to aggregate device from IOR and adds itself to the parent device using the <i>addDevice</i> operation?)  (Y/N/?)

## Test Summary - OE\_TC\_096

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0622 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.26. OE\_TC\_097 - Device :: Logical Devices - CF Interfaces

#### Test Case Number: OE\_TC\_097

Logical Device - CF Interfaces

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0623	The values associated with the parameters (PROFILE_NAME, COMPOSITE_DEVICE_IOR, DEVICE_ID and DEVICE_LABEL) as described in 3.1.3.2.4.5 shall be used to set the Device's softwareProfile, compositeDevice, identifier, and label attributes, respectively

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	P 3-98, Sec 3.3.3; 3-52, Sec 3.1.3.2.4.5

#### Test Objective

This test case verifies OE0623. The objective of this test is to verify that the values associated with the parameters (PROFILE\_NAME, COMPOSITE\_DEVICE\_IOR, DEVICE\_ID and DEVICE\_LABEL) as described in 3.1.3.2.4.5 are used to set the Device's softwareProfile, compositeDevice, identifier, and label attributes, respectively. Since these parameters are execute operation parameters, the devices being tested must be executable devices.

#### Places to Verify

Executable Devices

#### IDL References

The parameters passed to an executable device can be considered a sequence of DataTypes. The CF.idl definition for DataType is:

```
struct DataType
{
    string id;
    any value;
};
typedef sequence <DataType> Properties;
```

## Preconditions

- The Domain profile files are available
- The Domain Profile tests have been passed
- The source code files are available.

## Test Description

Note: “Parameters” refers to the four parameters in the requirement, PROFILE\_NAME, COMPOSITE\_DEVICE\_IOR, DEVICE\_ID and DEVICE\_LABEL.

A. For each DCD file, determine the devices parameters as described in the DCD’s. (OE0623)

1. **Pass:** Devices and parameter values are found.
2. **Untested:** No DCD files are located.
3. **N/A:** Devices and parameter values are not found.

For each device found, perform steps B and C.

B. Locate the source code for the device. (OE0623)

1. **Pass:** The source code for the device is found.
2. **Fail:** No source code for the device is found.

C. Verify that the parameters are saved properly. (OE0623)

1. **Pass:** The parameters are saved properly.
2. **Fail:** The parameters are not saved properly.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_097				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. For each DCD file, determine the devices parameters as described in the DCD's. (OE0623)</b>				
1. Locate all files ending in ".dcd.xml" and record the name.	<b>Pass:</b> One or more files are found. (OE0623)  <b>Untested:</b> No DCD files are located. (OE0623)			
2. Locate each occurrence of the "usagename" element for a device in the DCD files and record the value of the usagename.	<b>Pass:</b> "usagename" elements are found. (OE0623)  <b>Untested:</b> No "usagename" elements are found. (OE0623)		Components other than devices may have a usagename element. Verify that the element applies to a device. This will be the value for the DEVICE_LABEL parameter. An example of a usagename element is:  <usagename>someDevice</usagename>	
3. Record the value of the componentinstantiation id associated with the usagename.	The value of the componentinstantiation id is recorded.		This will be the value for the DEVICE_ID parameter. The relation between the componentinstantiation and the usage name elements is:  <componentinstantiation id = "someDevice_InstantiationID"> <usagename>Device</usagename> ... </componentinstantiation>	
4. Record the value of the compositepartofdevice, if it is defined. This element only applies for aggregate/composite devices so may not be present.	The value of the compositepartofdevice is recorded. If not defined, record "NONE".		This will be the value for the COMPOSITE_DEVICE_IOR parameter. An example of a compositepartofdevice element is: <compositepartofdevice refid="DCE:12345678-9200-1000-8000-123456789abc"/>	

5. Record the componentfile refid.	The value of the componentfile refid is recorded.			
6. Locate the componentfile id that the refid from step 5 refers to and record the name of the localfile.	The componentfile id is found.		Normally this will be earlier in the file than the refid. This file should be an SPD.	
7. Open the SPD file recorded in step 6, and determine if the code type is executable and if it has an entrypoint. Record the entrypoint.	An entry point is found.  N/A: The code type is not executable or no entrypoint is found. (OE0623)		If the component is not executable or does not have an entrypoint, then the DeviceManager cannot pass the parameters to it.	
<b>B. (and C) For each device found, perform steps 8 thru 13. Use the second log table for the following steps.</b>				
<b>1. Locate the source code for the device (OE0623).</b>				
8. Locate the source code for the device and record the file name.	<b>Pass:</b> The source code is located. (OE0623)  <b>Fail:</b> No source code is found. (OE0623)		The “main” of the device is the code we are locating. The developer’s engineer will be of assistance in locating this code.	
<b>2. Verify that the parameters are saved properly (OE0623).</b>				
9. Locate the portion of the initialization code for the device that processes the input parameters passed to it by the o/s.	The initialization code is located.		In a Unix type system, this would be the code that processes the argv argument that the “main” operation receives.	
10. Verify that a parameter with an id of “PROFILE_NAME” is processed and that the value is saved.	<b>Pass:</b> The propvalue is saved. (OE0623)  <b>Fail:</b> The proper value is not saved. (OE0623)		These parameters are id and value pairs, so when a parameter with an id match is found, the next parameter in the list should be saved.	
11. If the COMPOSITE_DEVICE_IOR is not NONE, verify that a parameter with an id of	<b>Pass:</b> The propvalue is saved. (OE0623)		These parameters are id and value pairs, so when a parameter with an id match is found, the next parameter in the list should be saved. This parameter may not be processed as	



“COMPOSITE_DEVICE_IOR” is processed and that the value is saved.	<b>Fail:</b> The proper value is not saved. (OE0623)		input but be set at instantiation time if the device is not a child device.	
12. Verify that a parameter with an id of “DEVICE_ID” is processed and that the value is saved.	<b>Pass:</b> The proper value is saved. (OE0623) <b>Fail:</b> The proper value is not saved.		These parameters are id and value pairs, so when a parameter with an id match is found, the next parameter in the list should be saved.	
13. Verify that a parameter with an id of “DEVICE_LABEL” is processed and that the value is saved.	<b>Pass:</b> The proper value is saved. (OE0623) <b>Fail:</b> The proper value is not saved. (OE0623)		These parameters are id and value pairs, so when a parameter with an id match is found, the next parameter in the list should be saved.	
<b>End of Test</b>				

Test Recording Log – OE_TC_097					
Step 1: DCD file:					
Step 2 (usagename)	Step 3 (component instantiation id)	Step 4 (compositepartofdevice)	Step 5 (componentfileref refid)	Step 6 (localfile)	Step 7 (entrypoint)

Test Recording Log – OE_TC_097				
Step 8 (source code)	Step 10 (PROFILE_NAME)	Step 11 (COMPOSITE_DEVICE_IO)	Step 12 (DEVICE_ID)	Step 13 (DEVICE_LABEL)

## Test Summary OE\_TC\_097

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0623 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.27. OE\_TC\_098 - Device :: Logical Device allocation properties****Test Case Number:** OE\_TC\_098

Device::Logical Device

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0627	For each logical device, allocation properties shall be defined in its referenced SPD's property file.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2, 15 May 2006	Page 3-98, Section 3.3.4
SCA Appendix D: Domain Profile	Version 2.2.2, 15 May 2006	Pages D.2.1, D-4, D.4.1.1, D-19, D-20
SCA Appendix C: Core Framework IDL	Version 2.2.2, 15 May 2006	Page C-27
DCE UUID standard (OSF Distributed Computing Environment, DCE 1.1 Remote Procedure Call).		all

**Test Objective**

This test case verifies OE0627. The objective of this test is to verify that a logical device defines all of the allocation properties that it uses in property descriptor files that are referenced in the device's SPD file. An id of an allocation property should be a Distributed Computing Environment Universal Unique Identifier (DCE UUID). All allocation property ids defined by the logical device in the property descriptor files referenced by the SPD must be supported by the *allocateCapacity* and *deallocateCapacity* operations in the logical device. The *allocateCapacity* and *deallocateCapacity* operations must support additional allocation properties other than those listed in the device's Domain Profile files (SPD, PRF).

**Places to Verify**

Logical Devices

**IDL References****Operations**

boolean *allocateCapacity* (in CF::Properties capacities)  
raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);

void *deallocateCapacity* (in CF::Properties capacities)  
    raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);

## Preconditions

- All The Domain Profile files are available
- The Domain Profile tests are passed.
- The source code files of the Operating Environment are available.

## Test Description

A. Identify the logical devices of the Operating Environment and the location of their SPD files and source code files. (OE0627)

1. **Pass:** The devices of the Operating Environment are identified and their SPD files and source code files exist.
2. **Untested:** The Operating Environment provides devices but their SPD files or source code files are not provided.
3. **N/A:** The Operating Environment does not provide devices.

For each of the logical devices in the Operating Environment:

B. Identify the allocation properties of the device, which are defined in property descriptor files referenced by Software Package Descriptor (SPD) files and verify that they are in Distributed Computing Environment Universal Unique Identifier (DCE UUID) format. If there are no allocation properties referenced by the SPD; then only perform Test Descriptions C and F, the other Test Description steps are non-applicable.(OE0627)

1. Identify and list the allocation properties of the device, which are defined in property descriptor files referenced by SPD files. (OE0627)

- a. **Pass:** The allocation properties are defined in property descriptor files referenced by SPD files.
- b. **NA:** Allocation properties do not exist in the SPD's referenced property files.

2. Verify that the allocation property ids of the device found in the property descriptor files are Distributed Computing Environment Universal Unique Identifiers (DCE UUIDs). (OE0627)

- a. **Pass:** The device allocation property ids are DCE UUIDs.
- b. **Fail:** The device allocation property ids are not DCE UUIDs.
- c. **NA:** The property descriptor files do not list allocation properties.

C. Verify that the *allocateCapacity* and the *deallocateCapacity* operations exist in the device. (OE0627)

1. **Pass:** The *allocateCapacity* and *deallocateCapacity* operations exist in the device.
2. **Fail:** The *allocateCapacity* operation does not exist in the device.
3. **Fail:** The *deallocateCapacity* operation does not exist in the device.

- D. Verify that the *allocateCapacity* operation supports the allocation property ids defined by the device's property descriptor files. (OE0627)
1. **Pass:** The *allocateCapacity* operation supports the property ids defined in the property descriptor.
  2. **Fail:** The *allocateCapacity* operation does not support the property ids defined property descriptor files.
  3. **N/A:** There are no allocation property ids defined in the property descriptor files referenced by the device's SPD.
- E. Verify that the *deallocateCapacity* operation supports the allocation property ids defined in the device's property descriptor. (OE0627)
1. **Pass:** The *deallocateCapacity* operation supports the property ids defined in the property descriptor.
  2. **Fail:** The *deallocateCapacity* operation does not support the property ids defined in the property descriptor files.
  3. **N/A:** There are no allocation properties listed in the property descriptor files referenced by the device's SPD.
- F. Verify that no other allocation properties other than the ones listed from the Test Description B results are supported in the *allocateCapacity* and *deallocateCapacity* operations of the device. (OE0627)
1. **Pass:** The *allocateCapacity* and the *deallocateCapacity* operation only support property ids that are listed in the property descriptor file(s) for the device.
  2. **Fail:** Either the *allocateCapacity* or the *deallocateCapacity* operation support property ids that are not listed in the property descriptor file(s) of the device.
  3. **Fail:** Either the *allocateCapacity* or *deallocateCapacity* operation supports allocation property ids but no allocation properties are listed in the SPD's property descriptor files.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_098				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the logical devices of the Operating Environment and the location of their SPD files and source code files. (OE0627)</b>				
1. Examine the DCD files to determine the devices provided by the OE and record the SPD file associated with each device.	<p><b>Pass:</b> The devices of the Operating Environment are identified and their SPD files and source code files exist. (OE0627)</p> <p><b>N/A:</b> Devices are not provided for the OE. (OE0627)</p> <p><b>Untested:</b> Devices are provided for the OE but their SPD files cannot be located. (OE0627)</p>		<p>May need to consult with the developer for the location of the logical devices.</p> <p>A device's id and name are usually specified in the DCD file as in the following example:</p> <pre>&lt;componentplacement&gt;   &lt;componentfile refid="xxxx" /&gt;   &lt;componentinstantiation id="DCE:xxxx-yyyyy-zzzz"&gt;     &lt;usagename&gt;zzzz&lt;/usagename&gt;   &lt;/componentinstantiation&gt; &lt;/componentplacement&gt; &lt;componentfiles&gt;   &lt;componentfile id="xxxx" type="SPD"&gt;     &lt;localfile name=".../yyyyy.spd.xml" /&gt;   &lt;/componentfile&gt; &lt;/componentfiles&gt;</pre> <p>Using the clue of the component placement ref and the information spd file could also give additional information on how to locate the source code.</p>	
2. Locate the source code for the logical device and record the name of the file.	<b>Untested:</b> The source code for the devices is not located. (OE0627)			
<b>For each of the logical devices in the Operating Environment:</b> <b>B. Identify the allocation properties of the device, which are defined in property descriptor files referenced by Software Package Descriptor (SPD) files and verify that they are in Universal Unique Identifier (DCEUUID) format. If there are no allocation properties referenced by the SPD, then only perform Test Descriptions C and F; the other test description steps are non-applicable. (OE0627)</b> <b>1. Identify and list the allocation properties of the device, which are defined in property descriptor files referenced by SPD files. (OE0627)</b>				



OE_TC_098				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Identify the property descriptor file reference in the SPD file of the device. List the property descriptor files for the device.	<p><b>Pass:</b> The allocation properties are defined in property descriptor files referenced by SPD files. (OE0627)</p> <p><b>N/A:</b> The SPD file does not reference property descriptor files. (OE0627)</p>		<p>The following is an example of the way a PRF could be listed in an SPD file.</p> <pre>&lt;propertyfile&gt; &lt;localfile name="zzzzz.prf.xml"/&gt; &lt;/propertyfile&gt;</pre>	
4. Examine the property descriptor files and identify the allocation properties of the device. Create a list of all the allocation property ids found.	<p><b>Pass:</b> The allocation properties of the device are located in the property descriptor files. (OE0627)</p> <p><b>N/A:</b> There are no allocation properties listed in the property descriptor files of the device. (OE0627)</p>		<p>The following is an example, but not the only example, of an allocation property described in a PRF file.</p> <pre>&lt;simple id="DCE:xxxx-yyy-zzzz" type="ulong" name="XXXX" mode="readwrite"&gt; &lt;description&gt;Allocation property for the device&lt;/description&gt;&lt;value&gt;xxxx&lt;/value&gt; &lt;kind kindtype="allocation"/&gt; &lt;action type="XX"/&gt;&lt;/simple&gt;</pre>	
<b>B.2 Verify that the allocation property ids of the device found in the property descriptor files are Distributed Computing Environment Universal Unique Identifiers (DCEUUIDs). (OE0627)</b>				
5. Verify that the allocation property ids of the device are DCE UUIDs.	<p><b>Pass:</b> The allocation property ids are DCE UUIDs. (OE0627)</p> <p><b>Fail:</b> The allocation property ids are not DCE UUIDs. (OE0627)</p> <p><b>N/A:</b> The property descriptor files do not contain allocation properties. (OE0627)</p>		<p>The following is a description of the DCE UUID from the SCA v2.2.2, Appendix D, Domain Profile, and page D-4.</p> <p>“The DCE UUID is as defined by the DCE UUID standard (adopted by CORBA). The DCE UUID format starts with the characters "DCE:" and is followed by the printable form of the DCE UUID, a colon, and a decimal minor version number, for example: "DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1". The decimal minor version number is optional. The version attribute specifies the version of the component. The name attribute is a user-friendly label for the softpkg element.</p>	

OE_TC_098				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>C. Verify that the <i>allocateCapacity</i> and the <i>deallocateCapacity</i> operations exist in the device. (OE0627)</b>				
6. Verify in the source code of the device that the <i>allocateCapacity</i> operation exists.	<b>Pass:</b> The <i>allocateCapacity</i> operation exists. (OE0627)  <b>Fail:</b> The <i>allocateCapacity</i> operation does not exist. (OE0627)			
7. Verify in the source code of the device that the <i>deallocateCapacity</i> operation exists.	<b>Pass:</b> The <i>deallocateCapacity</i> operation exists. (OE0627)  <b>Fail:</b> The <i>deallocateCapacity</i> operation does not exist. (OE0627)			
<b>D. Verify that the <i>allocateCapacity</i> operation supports the allocation property ids defined by the device's property descriptor files. (OE0627)</b>				
8. Verify in the source code of the device that the allocation property ids listed from the results of are supported by the <i>allocateCapacity</i> operation.	<b>Pass:</b> The allocation property ids are supported by the <i>allocateCapacity</i> operation. (OE0627)  <b>Fail:</b> The allocation property ids are <i>not</i> supported by the <i>allocateCapacity</i> operation. (OE0627)  <b>N/A:</b> The property descriptor files do not contain allocation properties. (OE0627)			

OE_TC_098				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>E. Verify that the <i>deallocateCapacity</i> operation supports the allocation property ids defined by the device's property descriptor files. (OE0627)</b>				
9. Verify in the source code of the device that the allocation properties ids found in the previous step(s) are supported by the <i>deallocateCapacity</i> operation.	<b>Pass:</b> The allocation property ids are supported by the <i>deallocateCapacity</i> operation. (OE0627)  <b>Fail:</b> The allocation property ids are not supported by the <i>deallocateCapacity</i> operation. (OE0627)  <b>N/A:</b> The property descriptor files do not contain allocation properties. (OE0627)			
<b>F. Verify that no other allocation properties other than the ones from the Test Description B results are supported in the <i>allocateCapacity</i> and <i>deallocateCapacity</i> operations of the device. (OE0627)</b>				

OE_TC_098				
Steps	Expected Results	Actual Results	Comments	Test Result
10. Verify that no other allocation property ids found in the source code of the <i>allocateCapacity</i> operation are located in the list of property ids found in manual step 5.	<p><b>Pass:</b> The <i>allocateCapacity</i> operation only supports allocation property ids listed in the property descriptor files for the device. (OE0627)</p> <p><b>Fail:</b> The <i>allocateCapacity</i> operation supports property ids outside of those listed in the property descriptor files for the device. (OE0627)</p> <p><b>Fail:</b> The <i>allocateCapacity</i> operation supports allocation property ids but no allocation properties are listed in the SPD's PRF files. (OE0627)</p>			

OE_TC_098				
Steps	Expected Results	Actual Results	Comments	Test Result
11. Verify that no other allocation property ids found in the source code of the <i>deallocateCapacity</i> operation are located in the list of property ids found in manual step 5.	<p><b>Pass:</b> The <i>deallocateCapacity</i> operation only supports allocation property ids listed in the property descriptor files for the device. (OE0627)</p> <p><b>Fail:</b> The <i>deallocateCapacity</i> operation supports allocation property ids outside of those listed in the property descriptor files for the device. (OE0627)</p> <p><b>Fail:</b> The <i>deallocateCapacity</i> operation supports allocation property ids but no allocation properties are listed in the SPD's PRF files. (OE0627)</p>			
<b>End of Test</b>				

Test Recording Log OE_TC_098										
Step 1a (Device SPD file)	Step 2 (Source code file for device)	Step 3 (List of PRF files for device)	Step 4 (List of allocation property ids.)	Step 5 (Y/N?)	Step 6 (Y/N?)	Step 7 (Y/N?)	Step 8 (Y/N?)	Step 9 (Y/N?)	Step 10 (Y/N?)	Step 11 (Y/N?)

**Test Summary OE\_TC\_098**

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x) Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

**Overall Test Result (Pass, Fail, Untested, or N/A):**

OE0627 \_\_\_\_\_

**Failed Items (Section/Step Number):**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_**Date Tested:** \_\_\_\_\_**Witness:** \_\_\_\_\_

**B.3.28. OE\_TC\_118 - Device :: allocateCapacity's acceptable properties****Test Case Number:** OE\_TC\_118

Device::AllocateCapacity

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0730	The <i>allocateCapacity</i> operation shall only accept properties for the input capacities parameter which are <i>simple</i> properties whose <i>kindtype</i> is <i>allocation</i> and whose <i>action</i> element is <i>external</i> contained in the component's SPD.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-62, Section 3.1.3.3.1.5.1.3;
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-19 thru D-23, Section D.4.1.1

**Test Objective**

This test case verifies OE0730. The objective of this test is to verify that only properties described in the component's SPD with a *kindtype* of *allocation* and an *action* element of *external* are accepted by the *allocateCapacity* operation. Note that the properties described are actually contained in the PRF files to which the SPD points.

**Places to Verify**

Devices that have properties.

**IDL References****Data**

```
struct DataType
```

```
{  
    string id;  
    any value;  
};
```

```
typedef sequence <DataType> Properties;
```

```
CF::Properties capacities;
```

**Exceptions**

```
exception InvalidCapacity
```

```
{  
    string msg;
```



```
CF::Properties capacities;  
};
```

### Operations

```
boolean allocateCapacity ( in CF::Properties capacities )  
    raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);
```

### Preconditions

- The ApplicationFactory and device source code files are available.

### Test Description

- A. Find where the ApplicationFactory's *create* operation invokes the device's *allocateCapacity* operation. (OE0730)
  1. **Pass:** The *allocateCapacity* operation is invoked by the ApplicationFactory's *create* operation.
  2. **Fail:** The *allocateCapacity* operation is not invoked by the ApplicationFactory's *create* operation.
- B. Verify that the properties supplied to the *allocateCapacity* operation are properties with a *kindtype* of *allocation* and whose *action* element is external. (OE0730).
  1. **Pass:** The properties supplied to the *allocateCapacity* operation are properties with a *kindtype* of *allocation* and whose *action* element is external.
  2. **Fail:** The properties supplied to the *allocateCapacity* operation are properties with a *kindtype* of *allocation* and whose *action* element is not external.
  3. **Fail:** The properties supplied to the *allocateCapacity* operation are properties with a *kindtype* that is not *allocation*.
- C. Verify that the *allocateCapacity* operation only accepts properties with a *kindtype* of *allocation* and whose *action* element is external. (OE0730).
  1. **Pass:** The *allocateCapacity* operation will only accept the properties *kindtype* is *allocation* and whose *action* element is external.
  2. **Fail:** The *allocateCapacity* operation will not accept the properties *kindtype* is *allocation* and whose *action* element is external.
  3. **Fail:** The *allocateCapacity* operation will not accept the properties other than the ones whose *kindtype* is *allocation* and whose *action* element is external.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_118				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Find where the ApplicationFactory's <i>create</i> operation invokes the device's <i>allocateCapacity</i> operation. (OE0730)</b>				
1. Search the source code for invocations of the <i>allocateCapacity</i> operation and record the file names.	<b>Pass:</b> Invocations of the <i>allocateCapacity</i> operation are found. (OE0730)  <b>Fail:</b> Invocation of the <i>allocateCapacity</i> operation is not found. (OE0730)			
2. Determine and record the input properties parameter.	The input properties parameter is recorded.			
<b>B. Verify that the properties supplied to the <i>allocateCapacity</i> operation are properties with a <i>kindtype</i> of <i>allocation</i> and whose <i>action</i> element is <i>external</i>. (OE0730).</b>				

OE_TC_118				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify that the values for the input properties parameter are extracted from the xml files and are properties with a <i>kindtype</i> of <i>allocation</i> and whose <i>action</i> element is external.	<p><b>Pass:</b> The values for the input properties parameter are extracted from the xml files and are properties with a <i>kindtype</i> of <i>allocation</i> and whose <i>action</i> element is external. (OE0730)</p> <p><b>Fail:</b> The values for the input properties parameter are extracted from the xml files and are properties with a <i>kindtype</i> of <i>allocation</i> and whose <i>action</i> element are not external (OE0730)</p> <p><b>Fail:</b> The values for the input properties parameter are extracted from the xml files and are properties whose <i>kindtype</i> is not <i>allocation</i> (OE0730)</p>		This will involve tracing backwards thru the code to determine where the values originate.	
<b>C. Verify that the <i>allocateCapacity</i> operation only accepts properties with a <i>kindtype</i> of <i>allocation</i> and whose <i>action</i> element is external. (OE0730).</b>				
4. Locate the <i>allocateCapacity</i> operation for each device.	The implementation of the <i>allocateCapacity</i> operation is found.			

OE_TC_118				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify that the <i>allocateCapacity</i> operation will only accept properties whose <i>kindtype</i> is <i>allocation</i> and whose <i>action</i> element is external.	<b>Pass:</b> The <i>allocateCapacity</i> operation will only accept properties whose <i>kindtype</i> is <i>allocation</i> and whose <i>action</i> element is external. (OE0730)  <b>Fail:</b> The operation will accept properties whose <i>kindtype</i> is <i>allocation</i> and whose <i>action</i> element is not external. (OE0730)  <b>Fail:</b> The operation will accept properties whose <i>kindtype</i> is not <i>allocation</i> . (OE0730)			
<b>End of Test</b>				

Test Recording Log – OE_TC_118			
Step1 (filename )	Step 2 (properties parameter)	Step3 (valid input values)	Step5 (inputs verified)

## Test Summary OE\_TC\_118

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0730 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.29. OE\_TC\_130 - LoadableDevice :: load raises LoadFail

**Test Case Number:** OE\_TC\_130

LoadableDevice::load raises LoadFail

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0432	The <i>load</i> operation shall raise the LoadFail exception when an attempt to load the device is unsuccessful.
OE0424	The error number shall indicate a CF ErrorNumberType.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-69, Section 3.1.3.3.2.3.3; Page 3-69 thru 3-70, Section 3.1.3.3.2.5.1.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 to C-4, C-29, C-30

#### Test Objective

This test case verifies OE0424 and OE0432. The objective of this test is to verify that the LoadFail exception occurs when a *load* operation for a LoadableDevice fails. The LoadFail exception indicates that the *load* operation failed due to device dependent reasons. The LoadFail exception indicates that an error occurred during an attempt to load the device. OE0424 verifies that the exception includes an error number of the type ErrorNumberType.

#### Places to Verify

Loadable and Executable Devices

#### IDL References

##### Data

enum ErrorNumberType {

CF\_NOTSET, CF\_E2BIG, CF\_EACCES, CF\_EAGAIN, CF\_EBADF, CF\_EBADMSG, CF\_EBUSY, CF\_ECANCELED,  
CF\_ECHILD, CF\_EDEADLK, CF\_EDOM, CF\_EEXIST, CF\_EFAULT, CF\_EFBIG, CF\_EINPROGRESS, CF\_EINTR,  
CF\_EINVAL, CF\_EIO, CF\_EISDIR, CF\_EMFILE, CF\_EMLINK, CF\_MSGSIZE, CF\_ENAMETOOLONG, CF\_ENFILE,  
CF\_ENODEV, CF\_ENOENT, CF\_ENOEXEC, CF\_ENOLCK, CF\_ENOMEM, CF\_ENOSPC, CF\_ENOSYS, CF\_ENOTDIR,

---

CF\_ENOTEMPTY, CF\_ENOTSUP, CF\_ENOTTY, CF\_ENXIO, CF\_EPERM, CF\_EPIPE, CF\_ERANGE, CF\_EROFS, CF\_ESPIPE, CF\_ESRCH, CF\_ETIMEDOUT, CF\_EXDEV };

### Exceptions

```
exception LoadFail {  
    CF::ErrorNumberType errorNumber;  
    string msg; };
```

### Operations

```
void load ( in CF::FileSystem fs, in string fileName, in CF::LoadableDevice::LoadType loadKind )  
    raises (CF::Device::InvalidState, CF::LoadableDevice::InvalidLoadKind, CF::InvalidFileName, CF::LoadableDevice::LoadFail);
```

### Preconditions

- Test case OE\_TC\_119 has been completed successfully.
- The LoadableDevice and ExecutableDevice source code files are available.

### Test Description

- A. Get the list of devices generated by test case OE\_TC\_119. (OE0424, OE0432)
  1. **Untested:** There are no devices.
- B. For each device, locate the source code file for the *load* operation. (OE0432, OE0424)  
For each source file found in step B, perform the following steps:
- C. Verify that the LoadFail exception is raised when a load fails. (OE0432)
  1. **Pass:** The LoadFail exception is raised when a load fails.
  2. **Fail:** The LoadFail exception is not raised when a load fails.
- D. Verify that there is a valid error number with the exception. (OE0424)
  1. **Pass:** The error number is valid.
  2. **Fail:** The error number is not valid.
  3. **Fail:** There is no error number with the exception.



## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_130				
Steps	Expected Results	Actual Results	Comments	Test Result
A. Get the list of devices generated by Test Case OE_TC_119.(OE0424, OE0432)				
1. Locate the results of OE_TC_119 and get the device list.	Untested: There are no devices. (OE0424, OE0432)  End of Test.			
B. For each device locate the source code file for the load operation:(OE0424, OE0432)				
2. Locate all declarations of the load operation for this device.	Declarations of load operations are found.		The intent of these two steps is to ensure we find the implementation of the load operation for every device and to minimize the next steps by having to look at base class implementations of the load operation only once.	
3. Determine where the load operation is implemented for this device. Record the source code file name.	Implementations of load operations are found.			
For each source file found in step B, perform steps C thru E.				
C. Verify that the LoadFail exception is raised when the load of a device fails.(OE0432)				
4. Determine that LoadFail exception is raised when the device cannot be loaded.	Pass: The load operation raises the LoadFail exception when the device cannot be loaded. (OE0432)  Fail: The load operation does not raise the LoadFail exception when the device cannot be loaded. (OE0432)			
D. Verify that there is a valid error number with the exception.(OE0424)				

OE_TC_130				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	<b>Pass:</b> The exception includes an error number (errorNumber) of ErrorNumberType. (OE0424)  <b>Fail:</b> The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0424)			
<b>End of Test</b>				

Test Recording Log – OE_TC_130		
Step 1 (Devices found)		
Step 2 (source code files)	Step 4 (exception raised - Y/N?)	Step 5 (error is an ErrorNumberType - Y/N?)

## Test Summary OE\_TC\_130

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0432 \_\_\_\_\_

OE0424 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.30. OE\_TC\_133 - LoadableDevice :: load raises InvalidFileName****Test Case Number:** OE\_TC\_133

LoadableDevice::load raises InvalidFileName

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0431	The <i>load</i> operation shall raise the CF <i>InvalidFileName</i> exception when the file designated by the input filename parameter cannot be found.
OE0599	The error number shall indicate a CF <i>ErrorNumberType</i> value.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-70, Section 3.1.3.3.2.5.1.5 Page 3-93, Section 3.1.3.6.4
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 to C-4, C-30

**Test Objective**

This test case verifies OE0431 and OE0599. The objective of this test is to verify that the LoadableDevice *load* operation raises the *InvalidFileName* exception when the file designated by the input filename parameter cannot be found. Furthermore, verify the exception contains an error number of the type CF::ErrorNumberType.

**Places to Verify**

LoadableDevice and ExecutableDevice

**IDL References****Data**

enum ErrorNumberType {

CF\_NOTSET, CF\_E2BIG, CF\_EACCES, CF\_EAGAIN, CF\_EBADF, CF\_EBADMSG, CF\_EBUSY, CF\_ECANCELED,  
 CF\_ECHILD, CF\_EDEADLK, CF\_EDOM, CF\_EEXIST, CF\_EFAULT, CF\_EFBIG, CF\_EINPROGRESS, CF\_EINTR,  
 CF\_EINVAL, CF\_EIO, CF\_EISDIR, CF\_EMFILE, CF\_EMLINK, CF\_MSGSIZE, CF\_ENAMETOOLONG, CF\_ENFILE,  
 CF\_ENODEV, CF\_ENOENT, CF\_ENOEXEC, CF\_ENOLCK, CF\_ENOMEM, CF\_ENOSPC, CF\_ENOSYS, CF\_ENOTDIR,  
 CF\_ENOTEMPTY, CF\_ENOTSUP, CF\_ENOTTY, CF\_ENXIO, CF\_EPERM, CF\_EPIPE, CF\_ERANGE, CF\_EROFS, CF\_ESPIPE,  
 CF\_ESRCH, CF\_ETIMEDOUT, CF\_EXDEV };

**Exceptions**

exception InvalidFileName {

```
        CF::ErrorNumberType errorNumber;  
        string msg; };  
exception InvalidState { string msg; };  
exception InvalidLoadKind{ };  
exception LoadFail { ErrorNumberType errorNumber; string msg; };
```

### Operations

```
void load (      in CF::FileSystem fs,  
                in string fileName,  
                in CF::LoadableDevice::LoadType loadKind )  
    raises (CF::Device::InvalidState, CF::LoadableDevice::InvalidLoadKind, CF::InvalidFileName, CF::LoadableDevice::LoadFail);
```

### Preconditions

- The LoadableDevice and ExecutableDevice source code files are available.

### Test Description

A. Identify the source code files that implement the LoadableDevice and ExecutableDevice *load* operation. (OE0431, OE0599)

1. **Untested:** The LoadableDevice and ExecutableDevice *load* operation source code files are not available.

For each LoadableDevice and ExecutableDevice *load* operation found within the OE under test do the following steps

- B. Verify that the *load* operation raises the CF::*InvalidFileName* when the file designated by the input filename parameter cannot be found. (OE0431).
1. **Pass:** The *load* operation raises the *InvalidFileName* exception when the file designated by the input filename parameter cannot be found.
  2. **Fail:** The *load* operation does not raise the *InvalidFileName* exception when the file designated by the input filename parameter cannot be found.
- C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)
1. **Pass:** The *InvalidFileName* exception provides an error number with a CF::ErrorNumberType value for the error condition.
  2. **Fail:** The *InvalidFileName* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_133				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the source code files that implement the LoadableDevice and ExecutableDevice load operation. (OE0431, OE0599)</b>				
1. Perform a search for <i>load</i> operations on all LoadableDevice and ExecutableDevice source code provided by the developer.	The devices and directories are recorded.  <b>Untested:</b> No results from search. There is no implementation of the <i>load</i> operation found. (OE0431, OE0599)		May need the help of the software engineer to locate the source code files directories.	
2. Examine the source code files returned in Step 1 and search for <i>load</i> operation implementations. Record the file names.	The <i>load</i> operation for each implementation is identified and recorded.			
<b>For each LoadableDevice and ExecutableDevice <i>load</i> operation found within the OE under test do the following steps</b>				
<b>B. Verify that the <i>load</i> operation raises the CF::<i>InvalidFileName</i> when the file designated by the input filename parameter cannot be found. (OE0431)</b>				
3. Search within the <i>load</i> operation for where the CF:: <i>InvalidFileName</i> exception is mentioned.	Identified the source code where <i>load</i> operation raises the CF:: <i>InvalidFileName</i> .			
4. Determine that CF:: <i>InvalidFileName</i> is raised when the file designated by the input filename parameter cannot be found.	<b>Pass:</b> The <i>load</i> operation raises the CF:: <i>InvalidFileName</i> . (OE0431)  <b>Fail:</b> The <i>load</i> operation does not raise the CF:: <i>InvalidFileName</i> . (OE0431)			

OE_TC_133				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)</b>				
<b>5.</b> Determine that the exception includes an error number ( <i>errorNumber</i> ) of the type ErrorNumberType.	<b>Pass:</b> The exception includes an error number ( <i>errorNumber</i> ) of ErrorNumberType. (OE0599)  <b>Fail:</b> The exception does not include an error number ( <i>errorNumber</i> ) of ErrorNumberType. (OE0599)			
<b>End of Test</b>				



Test Recording Log – OE_TC_133				
Step 1 (Device and directory name location)	Step 2 (Source file name)	Step 4 (Exception raised) Y/N	Step 5 (Correct error number type) Y/N	Notes

## Test Summary OE\_TC\_133

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0431\_\_\_\_\_

OE0599\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.31. OE\_TC\_134 - LoadableDevice :: unload raises InvalidFileName****Test Case Number:** OE\_TC\_134

LoadableDevice::unload raises InvalidFileName

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0436	The <i>unload</i> operation shall raise the CF <i>InvalidFileName</i> exception when the file designated by the input filename parameter cannot be found.
OE0599	The error number shall indicate a CF <i>ErrorNumberType</i> value.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-70, Section 3.1.3.3.2.5.2.5 Page 3-93, Section 3.1.3.6.4
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 to C-4, C-30

**Test Objective**

This test case verifies OE0436 and OE0599. The objective of this test is to verify that the *LoadableDevice unload* operation raises the *InvalidFileName* exception when the file designated by the input filename parameter cannot be found. Furthermore, verify the exception contains an error number of the type *CF::ErrorNumberType*.

**Places to Verify***LoadableDevice* and *ExecutableDevice***IDL References****Data**enum *ErrorNumberType* {

CF\_NOTSET, CF\_E2BIG, CF\_EACCES, CF\_EAGAIN, CF\_EBADF, CF\_EBADMSG, CF\_EBUSY, CF\_ECANCELED,  
CF\_ECHILD, CF\_EDEADLK, CF\_EDOM, CF\_EEXIST, CF\_EFAULT, CF\_EFBIG, CF\_EINPROGRESS, CF\_EINTR,  
CF\_EINVAL, CF\_EIO, CF\_EISDIR, CF\_EMFILE, CF\_EMLINK, CF\_MSGSIZE, CF\_ENAMETOOLONG, CF\_ENFILE,  
CF\_ENODEV, CF\_ENOENT, CF\_ENOEXEC, CF\_ENOLCK, CF\_ENOMEM, CF\_ENOSPC, CF\_ENOSYS, CF\_ENOTDIR,

CF\_ENOTEMPTY, CF\_ENOTSUP, CF\_ENOTTY, CF\_ENXIO, CF\_EPERM, CF\_EPIPE, CF\_ERANGE, CF\_EROFS, CF\_ESPIPE, CF\_ESRCH, CF\_ETIMEDOUT, CF\_EXDEV };

### Exceptions

exception InvalidFileName {

    CF::ErrorNumberType errorNumber;

    string msg; };

exception InvalidState { string msg; };

### Operations

void unload ( in string fileName )

    raises (CF::Device::InvalidState, CF::InvalidFileName );

### Preconditions

- The LoadableDevice and ExecutableDevice source code files are available.

### Test Description

Note: ExecutableDevice inherits from LoadableDevice. So even though this test case talks about the LoadableDevice *unload* operation, it may be found in the ExecutableDevice source code.

A. Identify the source code files that implement the LoadableDevice *unload* operation. (OE0436, OE0599)

1. **Untested:** The LoadableDevice *unload* operation source code files are not available.

For each LoadableDevice *unload* operation found in the LoadableDevice and ExecutableDevice components within the OE under test do the following steps

B. Verify that the LoadableDevice *unload* operation raises the InvalidFileName when the file designated by the input filename parameter cannot be found. (OE0436).

1. **Pass:** The *unload* operation raises the *InvalidFileName* exception when the file designated by the input filename parameter cannot be found.
2. **Fail:** The *unload* operation does not raise the *InvalidFileName* exception when the file designated by the input filename parameter cannot be found.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)

1. **Pass:** The *InvalidFileName* exception provides an error number with a CF::ErrorNumberType value for the error condition.
2. **Fail:** The *InvalidFileName* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_134				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the source code files that implement the LoadableDevice unload operation. (OE0436, OE0599)</b>				
1. Perform a key word search for <i>unload</i> operation on all LoadableDevice and ExecutableDevice source code provided by the developer	The directories are recorded.  <b>Untested:</b> No results from keyword search. There is no implementation of the <i>unload</i> operation found. (OE0436, OE0599)		May need the help of the software engineer to locate the source code files directories.	
2. Examine the source code files returned in Step 1 and search for <i>unload</i> operation implementation. Record the file name.	The <i>unload</i> operation implementation for each LoadableDevice and ExecutableDevice is identified and recorded.			
<b>For each LoadableDevice <i>unload</i> operation found within the OE under test do the following steps</b>				
<b>B. Verify that the LoadableDevice <i>unload</i> operation raises the InvalidFileName when the file designated by the input filename parameter cannot be found. (OE0436)</b>				
3. Map the implementing source code for the LoadableDevice or ExecutableDevice to the Device it is related to.	Device is mapped to its respective implementation.		Something like the following:  LoadableDev1 Dev1unload.cpp LoadableDev2 Dev2unload.cpp ...	
4. Search within the <i>unload</i> operation for where the input fileName parameter is verified.	Identify the source code where the <i>unload</i> operation verifies the input fileName parameter.			

OE_TC_134				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Determine that <i>InvalidFileName</i> is raised when the file designated by the input filename parameter cannot be found.	<p><b>Pass:</b> The <i>unload</i> operation raises the <i>InvalidFileName</i> exception when the file designated by the input filename parameter cannot be found. (OE0436)</p> <p><b>Fail:</b> The <i>unload</i> operation does not raise the <i>InvalidFileName</i> exception when the file designated by the input filename parameter cannot be found. (OE0436)</p> <p><b>Fail:</b> The <i>unload</i> operation raises the <i>InvalidFileName</i> exception even though the file designated by the input filename parameter is found. (OE0436)</p>			
<b>C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)</b>				
6. Determine if the exception includes an error number ( <i>errorNumber</i> ) of the type ErrorNumberType.	<p><b>Pass:</b> The exception includes an error number (<i>errorNumber</i>) of ErrorNumberType. (OE0599)</p> <p><b>Fail:</b> The exception does not include an error number (<i>errorNumber</i>) of ErrorNumberType. (OE0599)</p>			
<b>End of Test</b>				

Test Recording Log – OE_TC_134				
Step 1 (Directory name location)	Step 2 (Source file name)	Step 4 (Exception raised) Y/N	Step 6 (Correct error number type) Y/N	Notes



## Test Summary OE\_TC\_134

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0436 \_\_\_\_\_

OE0599 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.32. OE\_TC\_135 - ExecutableDevice :: execute raises InvalidFileName****Test Case Number:** OE\_TC\_135

ExecutableDevice::execute

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0452	The <i>execute</i> operation shall raise the CF <i>InvalidFileName</i> exception when the file name indicated by the input name parameter does not exist for the device.
OE0599	The CF <i>InvalidFileName</i> exception indicates an invalid file name was passed to a file service operation. The error number shall indicate a CF <i>ErrorNumberType</i> value.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-73, Section 3.1.3.3.5.1.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3, C-4, C-32

**Test Objective**

This test case verifies OE0452 and OE0599. The objective of this test is to (1) verify that the *execute* operation raises the *InvalidFileName* exception when the file name indicated by the input name parameter does not exist for the device, (2) verify that the error number in the exception is of the type CF::ErrorNumberType.

**Places to Verify**

ExecutableDevices

**IDL References****Data**

```
enum ErrorNumberType {  
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,  
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,  
    CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_MSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,  
    CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,
```

---

```
CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS,  
CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV};
```

### Exceptions

```
exception InvalidFileName {  
    CF::ErrorNumberType errorNumber;  
    string msg;  
}
```

### Operations

```
CF::ExecutableDevice::ProcessID_Type execute (  
    in string name,  
    in CF::Properties options,  
    in CF::Properties parameters  
)  
    raises (CF::Device::InvalidState, CF::ExecutableDevice::InvalidFunction, CF::ExecutableDevice::InvalidParameters,  
    CF::ExecutableDevice::InvalidOptions, CF::InvalidFileName, CF::ExecutableDevice::ExecuteFail);
```

### Preconditions

- All source code for all of the ExecutableDevices in the OE is available.

### Test Description

A. Identify the source code files that implement the *execute* operation. (OE0452)

1. **Untested:** The source code files of the *execute* operation are not available.

For each *execute* operation found within the OE under test, perform the following steps:

- B. Verify that the *execute* operation raises the *CF::InvalidFileName* when the file name indicated by the input name parameter does not exist for the device. (OE0452)
1. **Pass:** The *execute* operation raises *CF::InvalidFileName*.
  2. **Fail:** The *execute* operation does not raise *CF::InvalidFileName*.
- C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0599)
1. **Pass:** The exception includes an error number (errorNumber) of *ErrorNumberType*.
  2. **Fail:** The exception does not include an error number (errorNumber) of *ErrorNumberType*.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_135				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the source code files that implement the <i>execute</i> operation. (OE0452)</b>				
1. Perform a keyword search for <i>execute</i> operation on all ExecutableDevice source code provided by the developer.	<b>Untested:</b> The source code files of the <i>execute</i> operation are not available. (OE0452)		May need the help of the software engineer to locate the source code files.	
2. Examine the source code files returned in Step 1 and search for the <i>execute</i> operation implementation. Record the file name.	The <i>execute</i> operation for each implementation is identified and recorded.			
<b>B. Verify that the <i>execute</i> operation raises the <i>CF::InvalidFileName</i> when the file name indicated by the input name parameter does not exist for the device. (OE0452)</b>				
3. Search within the <i>execute</i> operation for where the <i>CF::InvalidFileName</i> exception is mentioned.	Identified the <i>execute</i> operation source code where the <i>execute</i> operation raises the <i>CF::InvalidFileName</i> .			
4. Determine that <i>CF::InvalidFileName</i> is raised when the file name indicated by the input name parameter does not exist for the device.	<b>Pass:</b> The <i>execute</i> operation raises the <i>CF::InvalidFileName</i> . (OE0452)  <b>Fail:</b> The <i>execute</i> operation does not raise the <i>CF::InvalidFileName</i> . (OE0452)		The actual requirement says 'when the file name ....does not exist for the device', but the implication would be 'when the file name ... does not exist within the file system'.	
<b>C. Verify that the error number that accompanies the exception is a <i>CF::ErrorNumberType</i> value. (OE0599)</b>				

OE_TC_135				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Determine if the exception includes an error number ( <i>errorNumber</i> ) of the type <i>ErrorNumberType</i> .	<b>Pass:</b> The exception includes an error number ( <i>errorNumber</i> ) of <i>ErrorNumberType</i> . (OE0599)  <b>Fail:</b> The exception does not include an error number ( <i>errorNumber</i> ) of <i>ErrorNumberType</i> . (OE0599)			
<b>End of Test</b>				

Test Recording Log – OE_TC_135			
Step2 (source file name)	Step4 (exception raised – Y/N)	Step5 (error number – Y/N)	Notes

## Test Summary OE\_TC\_135

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0452 \_\_\_\_\_

OE0599 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.33. OE\_TC\_145 - ExecutableDevice :: terminate raises InvalidProcess****Test Case Number:** OE\_TC\_145

ExecutableDevice::terminate raises InvalidProcess

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0458	The <i>terminate</i> operation shall raise the <i>InvalidProcess</i> exception when the process Id does not exist for the device.
OE0438	The errorNumber parameter shall indicate a CF ErrorNumberType value.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-71, Section 3.1.3.3.3.1 Page 3-73, Section 3.1.3.3.5.2.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 and C-4; C-31

**Test Objective**

This test case verifies OE0458 and OE0438. The objective of this test is to (1) verify that the *terminate* operation raises the *InvalidProcess* exception when the process Id does not exist for the device. (2) verify that the error number in the exception is of the type CF::ErrorNumberType.

**Places to Verify**

Core Framework ExecutableDevices

**IDL References****Data**

```
enum ErrorNumberType {
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
    CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_MSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,
    CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,
    CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS,
    CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV};
```



## Exceptions

```
exception InvalidProcess {  
    ErrorNumberType errorNumber; string msg; };
```

## Operations

```
void terminate (  
    in CF::ExecutableDevice::ProcessID_Type processId  
)  
    raises (CF::ExecutableDevice::InvalidProcess,  
           CF::Device::InvalidState);
```

## Preconditions

- The source code for all instances of the ExecutableDevice interface is available.

## Test Description

A. Identify all of the ExecutableDevices' source code files that implement the *terminate* operation. (OE0458)

1. **Untested:** The ExecutableDevices' source code files containing the *terminate* operation are not available.

For the *terminate* operation found in each ExecutableDevice of the OE under test, perform the following steps:

B. Verify that the *terminate* operation raises the *CF::InvalidProcess* exception when the process Id does not exist for the device. (OE0458)

1. **Pass:** The *terminate* operation raises the *CF::InvalidProcess* exception when the process Id does not exist for the device.
2. **Fail:** The *terminate* operation does not raise the *CF::InvalidProcess* exception when the process Id does not exist for the device.

C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0438)

3. **Pass:** The exception includes an error number (errorNumber) of ErrorNumberType.
4. **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_145				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify all of the ExecutableDevices' source code files that implement the <i>terminate</i> operation. (OE0458)</b>				
1. Perform a keyword search for the <i>terminate</i> operation on all source code provided by the developer.	<b>Untested:</b> The source code files of the <i>terminate</i> operation are not available. (OE0458)		May need the help of the software engineer to locate the source code files.	
2. Examine the source code files returned in Step 1 and search for the implementation of the ExecutableDevices' <i>terminate</i> operations. Record the file name.	The <i>terminate</i> operation for each implementation is identified and recorded.			
<b>For the <i>terminate</i> operation found in each ExecutableDevice of the OE under test, perform the following steps:</b>				
<b>B. Verify that the <i>terminate</i> operation raises the <i>CF::InvalidProcess</i> exception when the processId does not exist for the device. (OE0458)</b>				
3. Search within the <i>terminate</i> operation for where the processId is verified. The nonexistent processId should lead to the raising of the <i>InvalidProcess</i> .	Identified the source code where the <i>CF::InvalidProcess</i> exception is mentioned.			

OE_TC_145				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Determine that the <i>CF::InvalidProcess</i> exception is raised when the processId does not exist for the device.	<p><b>Pass:</b> The <i>terminate</i> operation raises the <i>CF::InvalidProcess</i> exception when the processId does not exist for the device. (OE0458)</p> <p><b>Fail:</b> The <i>terminate</i> operation does not raise the <i>CF::InvalidProcess</i> exception when the processId does not exist for the device. (OE0458)</p>			
<b>C. Verify that the error number that accompanies the exception is a <i>CF::ErrorNumberType</i> value. (OE0438)</b>				
5. Determine if the exception includes an error number ( <i>errorNumber</i> ) of the type <i>ErrorNumberType</i> .	<p><b>Pass:</b> The exception includes an error number (<i>errorNumber</i>) of <i>ErrorNumberType</i>. (OE0438)</p> <p><b>Fail:</b> The exception does not include an error number (<i>errorNumber</i>) of <i>ErrorNumberType</i>. (OE0438)</p>			
<b>End of Test</b>				

Test Recording Log – OE_TC_145			
Step2 (source file name)	Step4 (exception raised – Y/N)	Step5 (error number – Y/N)	Notes

## Test Summary OE\_TC\_145

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0458 \_\_\_\_\_

OE0438 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.34. OE\_TC\_153 - LoadableDevice :: load****Test Case Number:** OE\_TC\_153

CF::LoadableDevice::load

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0731	Multiple loads of the same file as indicated by the input <i>fileName</i> parameter shall not result in an exception.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-69, Section 3.1.3.3.2.5.1.3 Pages 3-69, Section 3.1.3.3.2.5.1 Pages 3-70, Section 3.1.3.3.2.5.2

**Test Objective**

This test case verifies OE0731. The objective of this test is to verify that the result of multiple loads of the same input *fileName* parameter does not raise an exception.

**Places to Verify**

Source Code Files with the *LoadableDevice load* operation

**IDL References****Operations**

void load (in FileSystem fs, in string fileName, in LoadType loadKind)  
raises (InvalidState, InvalidLoadKind, InvalidFileName, LoadFail);

**Pre conditions**

- The source code files for components implementing the *LoadableDevice* and *Executable* interfaces are available.

## Test Description

- A. Locate the source code for the *load* operation. (OE0731)
  - 1. **Untested:** The source code for the *load* operations is not found.
- B. Verify that an exception is not raised because of multiple load requests for the same file as indicated by the input *fileName* parameter. (OE0731)
  - 1. **Pass:** There are no exceptions for multiple loads of the same file.
  - 2. **Fail:** There are exceptions for multiple loads of the same file.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_153				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate the source code for the <i>load</i> operation. (OE0731)</b>				
1. Locate all instances of the <i>load</i> operation and record the file name.	Source code for the <i>load</i> operation is found.  <b>Untested:</b> The source code for the <i>load</i> operations is not found. (OE0731)			
<b>B. Verify that an exception is not raised because of multiple load requests for the same file as indicated by the input <i>fileName</i> parameter. (OE0731)</b>				
2. Find where the <i>fileName</i> parameter is being validated.	<b>Pass:</b> <i>fileName</i> validation is found. (OE0731)  <b>Fail:</b> <i>fileName</i> validation is not found. (OE0731)			
3. Determine if an exception is raised when the input <i>fileName</i> parameter is shown to already have been loaded. It is likely no test of the <i>fileName</i> value will be performed for prior loading.	<b>Pass:</b> There is no exception raised for multiple loads of the same file. (OE0731)  <b>Fail:</b> There is an exception raised for multiple loads of the same file. (OE0731)			
<b>End of Test</b>				



Test Recording Log – OE_TC_153		
Step 1 (load operation file(s))	Step 2 (no exceptions for multiple loads) Y/N	Notes

## Test Summary OE\_TC\_153

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0731\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.35. OE\_TC\_218 - Device Attributes

#### Test Case Number: OE\_TC\_218

Device attributes

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0395	The readonly operationalState attribute shall contain the device's operational state (ENABLED or DISABLED).
OE0401	The readonly softwareProfile attribute shall contain a profile element (Profile Descriptor) with a file reference to the SPD file.
OE0403	The readonly label attribute shall contain the device's label.
OE0404	The readonly compositeDevice attribute shall contain the object reference of the aggregate device when this device is a parent device.
OE0729	The readonly compositeDevice attribute shall contain a nil CORBA object reference when this device is not a parent device.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 1-2, Section 1.3.1.1 Pages 3-61 thru 3-62, Section 3.1.3.3.1.4.3 thru 3.1.3.3.1.4.6
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-25 and C-26

#### Test Objective

This test case verifies OE0395, OE0401, OE0403, OE0404, and OE0729. The objective of this test is to verify a device's operationalState, softwareProfile, label, and compositeDevice read-only attributes. With regard to requirement OE0401, a discussion of absolute and relative pathnames may be found in the SCA section 1.3.1.1 on page 1-2.

#### Places to Verify

The source code for all of the SCA Devices within the OE under test.

#### IDL References

##### Data

```
enum OperationalType { ENABLED, DISABLED };
```

```
readonly attribute CF::Device::OperationalType operationalState;
```

readonly attribute string softwareProfile;

readonly attribute string label;

readonly attribute CF::AggregateDevice compositeDevice;

## Preconditions

- The source code for all of the SCA Devices and their attributes is available.

## Test Description

A. Locate all implementations of the Devices interface in the source code. (OE0395, OE0401, OE0403, OE0404, OE0729)

1. **Untested:** Implementation of devices is not found.

For each implementation of Device found in step A, perform the following:

B. Verify that the readonly attribute named “operationalState” contains the device’s operational state. (OE0395)

1. **Pass:** The readonly attribute named “operationalState” contains the device’s operational state.
2. **Fail:** The readonly attribute named “operationalState” is not of type OperationalType.
3. **Fail:** The readonly attribute named “operationalState” does not contain the device’s operational state.

C. Verify that the readonly attribute named “softwareProfile” is a path name for the application’s SPD file. (OE0401)

1. **Pass:** The readonly attribute named “softwareProfile” is a path name for the application’s SPD file.
2. **Fail:** The readonly attribute named “softwareProfile” is not a string.
3. **Fail:** The readonly attribute named “softwareProfile” does not contain a file reference to the SPD file for this device.
4. **Fail:** The name of the SPD file is not stored in the readonly attribute named “softwareProfile”.

D. Verify that the readonly attribute named “label” contains the device’s label. (OE0403)

1. **Pass:** The readonly attribute named “label” contains the device’s label.
2. **Fail:** The readonly attribute named “label” is not a string.
3. **Fail:** The readonly attribute named “label” does not contain the device’s label.

E. Verify that the readonly attribute named “compositeDevice” only contains the object reference of an aggregate device. (OE0404, OE0729)

1. **Pass:** If this device is a parent device, then the readonly attribute named “compositeDevice” contains the object reference of an aggregate device.
2. **Pass:** If this device is not a parent device, then the readonly attribute named “compositeDevice” contains a nil CORBA object reference.
3. **Fail:** The readonly attribute named “compositeDevice” is not a string.
4. **Fail:** The readonly attribute named “compositeDevice” does not exist.

5. **Fail:** If this device is a parent device, and the readonly attribute named “compositeDevice” does not contain the object reference of the aggregate device.
6. **Fail:** If this device is not a parent device, and the readonly attribute named “compositeDevice” does not contain a nil CORBA object reference.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_218				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all implementations of the Devices interface in the source code. (OE0395, OE0401, OE0403, OE0404, OE0729)</b>				
1. Locate all implementations of the Devices interface in the source code.	<b>Untested:</b> Implementation of devices is not found. (OE0395, OE0401, OE0403, OE0404, OE0729)			
<b>For each implementation of Device found in step A, perform the following:</b>				
<b>B. Verify that the readonly attribute named “operationalState” contains the device’s operational state. (OE0395)</b>				
2. Verify that there is a readonly attribute named “operationalState” that is an OperationalType variable.	<p><b>Pass:</b> A readonly OperationalType variable named “operationalState” exists. (OE0395)</p> <p><b>Fail:</b> An OperationalType variable named “operationalState” does not exist. (OE0395)</p> <p><b>Fail:</b> An OperationalType variable named “operationalState” is not readonly. (OE0395)</p>		The most likely place to find attribute definitions is in the “private” portion of the software code or the declaration of a class. The definition of OperationalType is part of the IDL and should be defined as an enum with the values ENABLE and DISABLE only.	

OE_TC_218				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Find routines in the Device interface that uses the “operationalState” attribute.	<b>Pass:</b> Routines are found that use the “operationalState” attribute. (OE0395)  <b>Fail:</b> No routines are found that use the “operationalState” attribute (OE0395)			
4. Verify that the type of the “operationalState” attribute used in the found routines is OperationalType.	<b>Pass:</b> The type of the “operationalState” attribute used in the found routines is OperationalType. (OE0395)  <b>Fail:</b> The type of the “operationalState” attribute used in the found routines is not OperationalType. (OE0395)		This confirms the type of the attribute.	
5. Verify, based on the found routines’ names and functions, that the attribute named “operationalState” contains the device’s operational state.	<b>Pass:</b> Based on the found routines’ names and functions, the OperationalType variable named “operationalState” contains the device’s operational state. (OE0395)  <b>Fail:</b> Based on the found routines’ names and functions, the OperationalType variable named “operationalState” does not contain the device’s operational state. (OE0395)			
<b>C. Verify that the readonly attribute named “softwareProfile” is a path name for the application’s SPD file. (OE0401)</b>				

OE_TC_218				
Steps	Expected Results	Actual Results	Comments	Test Result
6. Verify that there is a readonly attribute named “softwareProfile” that is a string variable.	<p><b>Pass:</b> A readonly string variable named “softwareProfile” exists. (OE0401)</p> <p><b>Fail:</b> A string variable named “softwareProfile” does not exist. (OE0401)</p> <p><b>Fail:</b> A string variable named “softwareProfile” is not readonly. (OE0401)</p>		In a class the value of a readonly attribute is stored and can be retrieved, but is usually only set at the time of the class construction.	
7. Find routines in the Device interface that uses the “softwareProfile” attribute.	<p><b>Pass:</b> Routines are found that use the “softwareProfile” attribute. (OE0401)</p> <p><b>Fail:</b> No routines are found that use the “softwareProfile” attribute (OE0401)</p>			
8. Verify that the type of the “softwareProfile” attribute used in the found routines is string.	<p><b>Pass:</b> The type of the “softwareProfile” attribute used in the found routines is string. (OE0401)</p> <p><b>Fail:</b> The type of the “operationalState” attribute used in the found routines is not string. (OE0401)</p>		This confirms the type of the attribute.	



OE_TC_218				
Steps	Expected Results	Actual Results	Comments	Test Result
9. Verify, based on the found routines' names and functions, that the attribute named "softwareProfile" contains a file reference to the SPD file for this device.	<p><b>Pass:</b> Based on the found routines' names and functions, the string variable named "softwareProfile" contains a file reference to the SPD file for this device. (OE0401)</p> <p><b>Fail:</b> Based on the found routines' names and functions, the string variable named "softwareProfile" does not contain a file reference to the SPD file for this device. (OE0401)</p>			
<b>D. Verify that the readonly attribute named "label" contains the device's label. (OE0403)</b>				
10. Verify that there is a readonly attribute named "label" that is a string variable.	<p><b>Pass:</b> A readonly string variable named "label" exists. (OE0403)</p> <p><b>Fail:</b> A string variable named "label" does not exist. (OE0403)</p> <p><b>Fail:</b> A string variable named "label" is not readonly. (OE0403)</p>			
11. Find routines in the Device interface that uses the "label" attribute.	<p><b>Pass:</b> Routines are found that use the "label" attribute. (OE0403)</p> <p><b>Fail:</b> No routines are found that use the "label" attribute (OE0403)</p>			
12. Verify that the type of the "label" attribute used in the found routines is string.	<p><b>Pass:</b> The type of the "label" attribute used in the found routines is string. (OE0403)</p> <p><b>Fail:</b> The type of the "label" attribute used in the found routines is not string. (OE0403)</p>		This confirms the type of the attribute.	

OE_TC_218				
Steps	Expected Results	Actual Results	Comments	Test Result
13. Verify, based on the found routines' names and functions, that the attribute named "label" contains the device's label.	<p><b>Pass:</b> Based on the found routines' names and functions, the string variable named "label" contains the device's label. (OE0403)</p> <p><b>Fail:</b> Based on the found routines' names and functions, string variable named "label" does not contain the device's label. (OE0403)</p>			
<b>E. Verify that the readonly attribute named "compositeDevice" only contains the object reference of an aggregate device. (OE0404, OE0729)</b>				
14. Verify that there is a readonly attribute named "compositeDevice" that is an object reference variable.	<p><b>Pass:</b> A readonly object reference variable named "compositeDevice" exists. (OE0404)</p> <p><b>Fail:</b> An object reference variable named "compositeDevice" does not exist. (OE0404)</p> <p><b>Fail:</b> An object reference variable named "compositeDevice" is not readonly. (OE0404)</p>			
15. Find routines in the Device interface that uses the "compositeDevice" attribute.	<p><b>Pass:</b> Routines are found that use the "compositeDevice" attribute. (OE0404)</p> <p><b>Fail:</b> No routines are found that use the "compositeDevice" attribute (OE0404)</p>			

OE_TC_218				
Steps	Expected Results	Actual Results	Comments	Test Result
16. Verify that the type of the “compositeDevice” attribute used in the found routines is an object reference variable.	<b>Pass:</b> The type of the “compositeDevice” attribute used in the found routines is an object reference variable. (OE0404)  <b>Fail:</b> The type of the “compositeDevice” attribute used in the found routines is not an object reference variable. (OE0404)		This confirms the type of the attribute.	
17. Verify, based on the found routines’ names and functions, that the attribute named “compositeDevice” contains the object reference of the aggregate device if this device is the parent device.	<b>If this device is a parent device:</b> <b>Pass:</b> Based on the found routines’ names and functions, an object reference variable named “compositeDevice” contains the object reference of the aggregate device. (OE0404)  <b>Fail:</b> Based on the found routines’ names and functions, an object reference variable named “compositeDevice” does not contain the object reference of the aggregate device. (OE0404)			

OE_TC_218				
Steps	Expected Results	Actual Results	Comments	Test Result
18. Verify, based on the found routines' names and functions, that the attribute named "compositeDevice" contains a nil CORBA object reference if this device is not the parent device.	<b>If this is not a parent device:</b> <b>Pass:</b> Based on the found routines' names and functions, an object reference variable named "compositeDevice" contains a nil CORBA object reference (OE0729)  <b>Fail:</b> Based on the found routines' names and functions, an object reference variable named "compositeDevice" does not contain a nil CORBA object reference. (OE0729)			
<b>End of Test</b>				

Test Recording Log – OE_TC_218					
Step1 (Device implementations)					
Step2 & 5 (readonly OperationalType attribute named “operationalState”)	Step6 to 8 (readonly string attribute named “softwareProfile”)	Step9 (attribute “softwareProfile” is file ref to SPD file)	Step11 & 14 (readonly string attribute named “label” that is the device’s)	Step15 & 18 (non-nil readonly obj ref attribute named “compositeDevice”, if device is a parent device)	Step19 (attribute “compositeDevice” is nil, if device is not a parent device)

## Test Summary OE\_TC\_218

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0395\_\_\_\_\_

OE0401\_\_\_\_\_

OE0403\_\_\_\_\_

OE0404\_\_\_\_\_

OE0729\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### B.3.36. OE\_TC\_221 - ExecutableDevice Types

#### Test Case Number: OE\_TC\_221

ExecutableDevice Types

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0442	The value for a stack size shall be an unsigned long.
OE0443	The value for a priority shall be an unsigned long.

#### References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-72, Section 3.1.3.3.3.3.6, 3.1.3.3.3.7
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-31 and C-32

#### Test Objective

This test verifies requirements OE0442 and OE0443. The objective of this test case is to confirm that the *execute* method of the ExecutableDevice interface enforces the types of the STACK\_SIZE and PRIORITY contained in the “options” parameter.

#### Places to Verify

ExecutableDevices

#### IDL References

##### Data

```
constant string STACK_SIZE_ID = “STACK_SIZE”;  
constant string PRIORITY_ID = “PRIORITY”;
```

##### Operation

```
CF::ExecutableDevice::ProcessID_Type execute (in string name, in CF::Properties options, in CF::Properties parameters)  
    raises ( CF::Device::InvalidState, CF::ExecutableDevice::InvalidFunction, CF::ExecutableDevice::InvalidParameters,  
            CF::ExecutableDevice::InvalidOptions, CF::InvalidFileName, CF::ExecutableDevice::ExecuteFail);
```

## Preconditions

- The ExecutableDevice source code files are available.

## Test Description

A. Locate all implementations of the *execute* operation for the ExecutableDevice interface. (OE0442, OE0443)

1. **Fail:** There is no implementation of the *execute* operation for ExecutableDevice.

For each occurrence of the *execute* operation perform the following:

B. Verify the value for a stack size is an unsigned long. (OE0442)

1. **Pass:** When an options parameter has an id that is “STACK\_SIZE” the value is an unsigned long.
2. **Fail:** When an options parameter has an id that is “STACK\_SIZE” the value is not an unsigned long

C. Verify the value for a priority is an unsigned long. (OE0443)

1. **Pass:** When an options parameter has an id that is “PRIORITY” the value is an unsigned long.
2. **Fail:** When an options parameter has an id that is “PRIORITY” the value is not an unsigned long.



## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_221				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all implementations of the <i>execute</i> operation for the ExecutableDevice interface. (OE0442, OE0443)</b>				
1. Record the names of the implementation files.	<b>Fail:</b> There is no implementation of the <i>execute</i> operation for ExecutableDevice. (OE0442, OE0443)			
<b>For each occurrence of the <i>execute</i> operation perform the following:</b>				
<b>B. Verify the value for a stack size is an unsigned long. (OE0442)</b>				
2. Verify the options parameter has an id that is "STACK_SIZE" then the value is an unsigned long.	<b>Pass:</b> The options parameter has an id that is "STACK_SIZE" the value is an unsigned long. (OE0442)  <b>Fail:</b> The options parameter has an id that is "STACK_SIZE" the value is not an unsigned long. (OE0442)		The "STACK_SIZE" is an optional parameter, but it must be properly handled when it is specified as a parameter to the execute method. (See OE0448)	
<b>C. Verify the value for a priority is an unsigned long. (OE0443)</b>				

OE_TC_221				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify the options parameter has an id that is "PRIORITY" and the value is an unsigned long.	<b>Pass:</b> The options parameter has an id that is "PRIORITY" the value is an unsigned long.(OE0443)  <b>Fail:</b> The options parameter has an id that is "PRIORITY" the value is not an unsigned long. (OE0443)		The "PRIORITY" is an optional parameter, but it must be properly handled when it is specified as a parameter to the execute method. (See OE0448)	
<b>End of Test</b>				

Test Recording Log – OE_TC_221		
Step 1 (file name)	Step 2 (stack size type)	Step 3 (priority type)

## Test Summary OE\_TC\_221

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0442 \_\_\_\_\_

OE0443 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.37. OE\_TC\_222 - ExecutableDevice :: execute raises exceptions****Test Case Number:** OE\_TC\_222

ExecutableDevice::execute raises exceptions

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0450	The <i>execute</i> operation shall raise the <i>InvalidState</i> exception if upon entry the device's <i>adminState</i> attribute is either <i>LOCKED</i> or <i>SHUTTING_DOWN</i> or its <i>operationalState</i> attribute is <i>DISABLED</i> .
OE0451	The <i>execute</i> operation shall raise the <i>InvalidFunction</i> exception when the function indicated by the input name parameter does not exist for the device.
OE0453	The <i>execute</i> operation shall raise the <i>InvalidParameters</i> exception when the input parameter ID or value attributes are not valid strings.
OE0454	The <i>execute</i> operation shall raise the <i>InvalidOptions</i> exception when the input options parameter does not comply with sections 3.1.3.3.3.6 <i>STACK_SIZE_ID</i> and 3.1.3.3.3.7 <i>PRIORITY_ID</i> .
OE0455	The <i>execute</i> operation shall raise the <i>ExecuteFail</i> exception when the operating system "execute" function for the device is not successful.
OE0444	The error number shall indicate a CF <i>ErrorNumberType</i> value.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-71 and 3-72, Section 3.1.3.3.3.3, Pages 3-73, Section 3.1.3.3.3.5.1.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-24, and C-30 to C-32

**Test Objective**

This test case verifies OE0450, OE0451, OE0453, OE0454, OE0455 and OE0444. The objective of this test is to verify that the Executable Device *execute* operation raises various exceptions based on the circumstances of a failed execution. It raises the *InvalidState* exception when at the beginning of the execution, the device's *adminState* attribute is *LOCKED* or *SHUTTING\_DOWN* or its *operationalState* attribute is *DISABLED*. It raises the *InvalidParameters* exception when the input parameter ID or value attributes are not valid strings. It raises the *InvalidOptions* exception when the input options parameter does not comply with sections that describe *STACK\_SIZE\_ID* and *PRIORITY\_ID*. It raises the *InvalidFunction* exception when the function indicated by the input name parameter does not exist for the device. Finally, it raises the *ExecuteFail* exception when the operating system "execute" function for the device is not successful. Furthermore, the test case verifies that the *ExecuteFail* exception contains an error number of the type CF::ErrorNumberType.

## Places to Verify

ExecutableDevices

## IDL References

### Data

```
enum ErrorNumberType {  
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,  
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,  
    CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_MSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,  
    CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,  
    CF_NOTEMPTY, CF_NOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS,  
    CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };  
  
const string STACK_SIZE_ID = "STACK_SIZE";  
  
const string PRIORITY_ID = "PRIORITY";
```

### Exceptions

```
exception InvalidState { string msg; };  
  
exception InvalidFunction{};  
  
exception InvalidParameters { Properties invalidParms; };  
  
exception InvalidOptions { Properties invalidOpts; };  
  
exception ExecuteFail { ErrorNumberType errorNumber; string msg; };
```

### Operations

```
ProcessID_Type execute (in string name, in Properties options, in Properties parameters)  
    raises (InvalidState, InvalidFunction, InvalidParameters, InvalidOptions, InvalidFileName, ExecuteFail);
```

## Preconditions

- The ExecutableDevice source code files are available.

## Test Description

- A. Locate all implementations of the execute operation of the ExecutableDevice interface. (OE0450, OE0451, OE0453, OE0454, OE0455, OE0444)
1. **Untested:** There is no source code for the *execute* operation.
  2. **Pass:** Source code for the *execute* operation is found
- For each implementations of the *execute* operation of the ExecutableDevice interface found, perform the following:
- B. When at the beginning of execution, the device's adminState attribute is LOCKED or SHUTTING\_DOWN or its operationalState attribute is DISABLED, verify that the *InvalidState* exception is raised. (OE0450)
1. **Pass:** The *InvalidState* exception is raised when the device's adminState attribute is LOCKED or SHUTTING\_DOWN or its operationalState attribute is DISABLED.
  2. **Fail:** The *InvalidState* exception is not raised when the device's adminState attribute is LOCKED or SHUTTING\_DOWN or its operationalState attribute is DISABLED.
- C. Under the condition when the input parameter ID or value attributes are not valid strings, verify the *InvalidParameters* exception is raised and it contains a list of invalid parameters. (OE0453)
1. **Pass:** The *InvalidParameters* exception is raised when the input parameter name or the input parameters contents is not a valid string.
  2. **Fail:** The *InvalidParameters* exception is not raised when the input parameter name or the input parameters contents is not a valid string.
- D. When the input options parameter does not comply with section 3.1.3.3.3.6 and 3.1.3.3.3.7 for the stack size and priority parameters, verify that the *InvalidOptions* exception is raised containing a list of invalid options. (OE0454)
1. **Pass:** The *InvalidOptions* exception is raised when the input options named "STACK\_SIZE" or "PRIORITY" are not unsigned long.
  2. **Fail:** The *InvalidOptions* exception is not raised when, the input options named "STACK\_SIZE" or "PRIORITY" are invalid and are not unsigned longs.
- E. When the function indicated by the input name parameter does not exist for the device, verify that the *InvalidFunction* exception is raised. (OE0451)
1. **Pass:** The *InvalidFunction* exception is raised when the function name does not exist for the device.
  2. **Fail:** The *InvalidFunction* exception is not raised when the function name does not exist for the device.
- F. Verify that if the OS "execution" command is not successful, then the *ExecuteFail* exception is raised and it contains a proper error number.
1. Verify that if the OS "execution" command is not successful the *ExecuteFail* exception is raised. (OE0455)
    - a. **Pass:** The *ExecuteFail* exception is raised properly.
    - b. **Fail:** The OS "execute" operation fails and the *ExecuteFail* exception is not raised.
    - c. **Fail:** The OS "execute" operation succeeds and the *ExecuteFail* exception is raised.
  2. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0444)

- a. **Pass:** The *ExecuteFail* exception provides an error number with a CF::ErrorNumberType value for the error condition.
- b. **Fail:** The *ExecuteFail* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.



## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_222				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all implementations of the execute operation of the ExecutableDevice interface. (OE0450, OE0451, OE0453, OE0454, OE0455, OE0444)</b>				
1. Perform a keyword search for <i>execute</i> operation on all ExecutableDevice source code provided by the developer. Examine the source code files returned and search for <i>execute</i> operation implementation. Record file names of the source code where <i>execute</i> operation implementations were found.	The directories and files are recorded.  <b>Untested:</b> No results from keyword search. There is no implementation of the ExecutableDevice <i>execute</i> operation found. (OE0450, OE0451, OE0452, OE0454, OE0455, OE0444)		May need the help of the software engineer to locate the source code files directories.	
<b>For each implementations of the <i>execute</i> operation of the ExecutableDevice interface found, perform the following:</b>				
<b>B. When at the beginning of execution, the device's adminState attribute is LOCKED or SHUTTING_DOWN or its operationalState attribute is DISABLED, the InvalidState exception is raised. (OE0450)</b>				
2. Identify within the <i>execute</i> operation where the device's adminState and operationalState attributes are validated for the <i>execute</i> operation.	<b>Pass:</b> Validation of the adminState and the operationalState attributes was found. (OE0450)  <b>Fail:</b> No validation of the adminState and the operationalState attributes can be found. (OE0450)			

OE_TC_222				
Steps	Expected Results	Actual Results	Comments	Test Result
3. When the device's adminState is LOCKED or SHUTTING_DOWN, verify that the <i>InvalidState</i> exception is raised.	<p><b>Pass:</b> The <i>InvalidState</i> exception is raised when the device's adminState is LOCKED or SHUTTING_DOWN. (OE0450)</p> <p><b>Fail:</b> The <i>InvalidState</i> exception is not raised when the device's adminState is LOCKED or SHUTTING_DOWN. (OE0450)</p>			
4. When the device's operationalState is DISABLED, verify that the <i>InvalidState</i> exception is raised.	<p><b>Pass:</b> The <i>InvalidState</i> exception is raised when the device's operationalState is DISABLED. (OE0450)</p> <p><b>Fail:</b> The <i>InvalidState</i> exception is not raised when the device's operationalState is DISABLED. (OE0450)</p>			
<b>C. Verify that when the input parameter ID or value attributes are not valid strings, the <i>InvalidParameters</i> exception is raised and it contains a list of invalid parameters. (OE0453)</b>				
5. Identify where the input attributes name and parameters are validated for the device.	<p><b>Pass:</b> Validation of the input parameters was found. (OE0453)</p> <p><b>Fail:</b> No validation of the input parameters can be found. (OE0453)</p>			

OE_TC_222				
Steps	Expected Results	Actual Results	Comments	Test Result
6. When the input parameter <i>name</i> (ID) is not a valid string, verify that the <i>InvalidParameters</i> exception is raised.	<p><b>Pass:</b> The <i>InvalidParameters</i> exception is raised when the input parameter <i>name</i> is not a valid string. (OE0453)</p> <p><b>Fail:</b> The <i>InvalidParameters</i> exception is not raised when the input parameter <i>name</i> is not a valid string. (OE0453)</p>			
7. When the input <i>parameters</i> contents are not valid strings, verify that the <i>InvalidParameters</i> exception is raised.	<p><b>Pass:</b> The <i>InvalidParameters</i> exception is raised when the input <i>parameters</i> contents are not valid strings. (OE0453)</p> <p><b>Fail:</b> The <i>InvalidParameters</i> exception is not raised when the input <i>parameters</i> contents are not valid strings. (OE0453)</p>			
<b>D. When the input options parameter does not comply with section 3.1.3.3.3.6 and 3.1.3.3.3.7 for the stack size and priority parameters, verify that the <i>InvalidOptions</i> exception is raised containing a list of invalid options. (OE0454)</b>				
8. Identify where the input option parameters are validated for the device.	<p><b>Pass:</b> Validation of the input option parameters was found. (OE0454)</p> <p><b>Fail:</b> No validation of the input option parameters can be found. (OE0454)</p>			

OE_TC_222				
Steps	Expected Results	Actual Results	Comments	Test Result
9. When the input option named "STACK_SIZE" is not an unsigned long, verify that the <i>InvalidOptions</i> exception is raised.	<b>Pass:</b> The <i>InvalidOptions</i> exception is raised when the input option named "STACK_SIZE" is not an unsigned long. (OE0454)  <b>Fail:</b> The <i>InvalidOptions</i> exception is not raised when the input option named "STACK_SIZE" is not an unsigned long. (OE0454)			
10. When the input option named "PRIORITY" is not an unsigned long, verify that the <i>InvalidOptions</i> exception is raised.	<b>Pass:</b> The <i>InvalidOptions</i> exception is raised when the input option named "PRIORITY" is not an unsigned long. (OE0454)  <b>Fail:</b> The <i>InvalidOptions</i> exception is not raised when the input option named "PRIORITY" is not an unsigned long. (OE0454)			
<b>E. When the function indicated by the input name parameter does not exist for the device, verify that the <i>InvalidFunction</i> exception is raised. (OE0451)</b>				
11. Identify where the function name (input name parameter) is validated for the device.	<b>Pass:</b> Validation of the function name was found. (OE0451)  <b>Fail:</b> No validation of the function name can be found. (OE0451)			

OE_TC_222				
Steps	Expected Results	Actual Results	Comments	Test Result
12. When the function name does not exist for the device, verify that the <i>InvalidFunction</i> exception is raised.	<p><b>Pass:</b> The <i>InvalidFunction</i> exception is raised when the function name does not exist for the device. (OE0451)</p> <p><b>Fail:</b> The <i>InvalidFunction</i> exception is not raised when the function name does not exist for the device. (OE0451)</p>			
<b>F. Verify that if the OS “execution” command is not successful, then the <i>ExecuteFail</i> exception is raised and it contains a proper error number.</b>				
<b>1. Verify that if the OS “execution” command is not successful, the <i>ExecuteFail</i> exception is raised. (OE0455)</b>				
13. Identify where the “execution” command is not successful for the <i>execute</i> operation.	<p><b>Pass:</b> The place where the command failed to execute was found. (OE0455)</p> <p><b>Fail:</b> No place, where it “executes” the command can be found. (OE0455)</p>			
14. If the OS “execution” command is not successful, verify that the <i>ExecuteFail</i> exception is raised.	<p><b>Pass:</b> The <i>ExecuteFail</i> exception is raised if the OS “execution” command is not successful. (OE0455)</p> <p><b>Fail:</b> The <i>ExecuteFail</i> exception is not raised if the OS “execution” command is not successful. (OE0455)</p> <p><b>Fail:</b> The <i>ExecuteFail</i> exception is raised if the OS “execution” command is successful. (OE0455)</p>			

OE_TC_222				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>2. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0444)</b>				
15. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	<b>Pass:</b> The exception includes an error number (errorNumber) of ErrorNumberType. (OE0444)  <b>Fail:</b> The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0444)			
<b>End of Test</b>				

Test Recording Log – OE_TC_222					
Step1 (file & directory for <i>execute</i> operation implementation)					
Step2-4 ( <i>InvalidState</i> for executable device)	Step5-7 ( <i>InvalidParameters</i> when name and parameters are not strings)	Step8-10 ( <i>InvalidOptions</i> when options are invalid)	Step11-12 ( <i>InvalidFunction</i> when function name is bad)	Step13-14 ( <i>ExecuteFail</i> when execution is not successful)	Step15 (Error number is ErrorNumberType)

## Test Summary OE\_TC\_222

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0450 \_\_\_\_\_

OE0451 \_\_\_\_\_

OE0453 \_\_\_\_\_

OE0454 \_\_\_\_\_

OE0455 \_\_\_\_\_

OE0444 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_



**B.3.38. OE\_TC\_227 - ExecutableDevice :: execute****Test Case Number:** OE\_TC\_227

ExecutableDevice::execute

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0445	The <i>execute</i> operation shall execute the function or file identified by the input name parameter using the input parameters and options parameters.
OE0446	The <i>execute</i> operation shall convert the input parameters (id/value string pairs) parameter to the standard argv of the POSIX exec family of functions, where argv(0) is the function name.
OE0447	The <i>execute</i> operation shall map the input parameters parameter to argv starting at index 1 as follows, argv (1) maps to input parameters (0) id and argv (2) maps to input parameters (0) value and so forth.
OE0448	The <i>execute</i> operation shall use these options, when specified, to set the operating system's process/thread stack size and priority, for the executable image of the given input name parameter.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-72 through 3-73, Section 3.1.3.3.5.1.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-2 Pages C-31 and C-32
IEEE Standard for Information Technology – Standardized Application Environment Profile (AEP) – POSIX® Realtime and Embedded Application Support, IEEE Std 1003.13-2003.	10 September 2004	All

**Test Objective**

This test case verifies OE0445, OE0446, OE0447 and OE0448. The objective of this test is to verify that the Executable Device *execute* operation executes the entity identified by the *id* parameter using the given *parameters* and *options* parameters. This test is to verify the proper parsing of its *parameters* parameter and its *options* parameter. It also will verify that should the *options* parameters include *STACK\_SIZE\_ID* and *PRIORITY\_ID*, they will be used to set the OS' process/thread stack size and priority.

**Places to Verify**

Executable Devices source code

## IDL References

### Data

```
const string STACK_SIZE_ID = "STACK_SIZE";
```

```
const string PRIORITY_ID = "PRIORITY";
```

```
struct DataType {  
    string id;  
    any value; };
```

```
typedef sequence <DataType> Properties;
```

```
typedef long ProcessID_Type;
```

### Operations

```
CF::ExecutableDevice::ProcessID_Type execute (  
    in string name,  
    in CF::Properties options,  
    in CF::Properties parameters)  
    raises ( CF::Device::InvalidState, CF::ExecutableDevice::InvalidFunction, CF::ExecutableDevice::InvalidParameters,  
            CF::ExecutableDevice::InvalidOptions, CF::InvalidFileName, CF::ExecutableDevice::ExecuteFail);
```

## Preconditions

- The ExecutableDevice interface source code files are available.

## Test Description

A. Locate all implementations of the ExecutableDevice interface. (OE0445, OE0446, OE0447, OE0448)

1. N/A: There are no implementations of the ExecutableDevice interface.

For each implementation of the ExecutableDevice interface, locate the code that implements the *execute* operation of a device and perform the following:

B. Verify that the input parameters are converted to the standard argv of the POSIX family of functions. (OE0445, OE0446)

1. **Pass:** The input parameters are converted to the standard argv of the POSIX family of functions.
  2. **Fail:** argv (0) is not the function name.
  3. **Fail:** The input parameters are not converted to the standards argv family of arguments.
- C. Verify that the input *parameters* are mapped to argv starting at index 1 as follows: input *parameters* (0) id maps to argv (1) and input *parameters* (0) value maps to argv (2) and so forth. (OE0445, OE0447)
- Note: A fuller mapping of parameters are *parameters*(0) id/value pair are mapped to argv(1) & (2) respectively,  
*parameters*(1) id/value pair are mapped to argv(3) & (4) respectively,  
*parameters*(2) id/value pair are mapped to argv(5) & (6) respectively, and so on.
1. **Pass:** The device input *parameters* are mapped to argv as stated above.
  2. **Fail:** The *parameters*' id is not mapped to an odd argv parameter, starting with argv (1).
  3. **Fail:** The *parameters*' value does not map to an even argv parameter, starting with argv (2).
- D. Verify that the *options*, when specified, are used to set stack size and priority properly. (OE0445, OE0448)
1. **Pass:** The stack size and priority are set properly by corresponding *options* values.
  2. **Fail:** The stack size is not set in accordance with the input *options*.
  3. **Fail:** The priority is not set in accordance with the input *options*.
- E. Verify that the ExecutableDevice executes the function or file identified by the input *name* parameter. (OE0445)
1. **Pass:** The named function or file is executed.
  2. **Fail:** The named function or file is not executed.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_227				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all implementations of the ExecutableDevice interface. (OE0445, OE0446, OE0447, OE0448)</b>				
1. Locate all implementations of the ExecutableDevice interface in the source code.	<p><b>N/A:</b> No ExecutableDevice source code found. (OE0445, OE0446, OE0447, OE0448)</p> <p><b>Pass:</b> ExecutableDevice source code is found. (OE0445, OE0446, OE0447, OE0448)</p>			
<b>For each implementation of the ExecutableDevice interface, locate the code that implements the <i>execute</i> operation of a device and perform the following:</b>				
<b>B. Verify that the input parameters are converted to the standard argv of the POSIX family of functions. (OE0445, OE0446)</b>				
2. Verify that the input parameters are converted to the standard argv of the POSIX family of functions.	<p><b>Pass:</b> The input parameters are converted to the standard argv of the POSIX family of functions. (OE0445, OE0446)</p> <p><b>Fail:</b> The input parameters are not converted to the standard argv of the POSIX family of functions. (OE0445, OE0446)</p>		All of the steps are required to verify OE0445. That requirement is about the execute operation being successful. If any of the other requirements fail then it can be assumed that the execute operation cannot be performed successfully.	
3. Verify that the argv(0) parameter is not used as part of an id/value string pairs.	<p><b>Pass:</b> The argv(0) parameter is not used as part of an id/value string pairs. (OE0445, OE0446)</p> <p><b>Fail:</b> The argv(0) parameter is used as part of an id/value string pairs. (OE0445, OE0446)</p>			

OE_TC_227				
Steps	Expected Results	Actual Results	Comments	Test Result
C. Verify that the input <i>parameters</i> are mapped to argvs starting at index 1 as follows: input <i>parameters</i> (0) id maps to argv(1) and input <i>parameters</i> (0) value maps to argv(2) and so forth. (OE0445, OE0447)				
4. Starting with <i>parameters</i> (0) and continuing with id fields of the id/value pair, verify that they are always mapped to odd numbered argv items.	<p><b>Pass:</b> The <i>parameters</i> id of an id/value pair is always mapped to the odd numbered argv items. (OE0445, OE0457)</p> <p><b>Fail:</b> The <i>parameters</i> id of an id/value pair is mapped to an even numbered argv items. (OE0445, OE0457)</p>		The argv index for the parameters id of the id/value pair s should be (parameter_index*2) +1. For example parameters indexid → argv 0 (0*2)+1=1 1 (1*2)+1=3 2 (2*2)+1=5 3 (3*2)+1=7	
5. Starting with <i>parameters</i> (0) and continuing with value fields of the id/value pair, verify that they are always mapped to even numbered argv items.	<p><b>Pass:</b> The <i>parameters</i> value of an id/value pair is always mapped to the even numbered argv items. (OE0445, OE0457)</p> <p><b>Fail:</b> The <i>parameters</i> value of an id/value pair is mapped to an odd numbered argv items. (OE0445, OE0457)</p>		The argv index for the parameters value of the id/value pair s should be (parameter_index*2) +2. For example parameters indexvalue → argv 0 (0*2)+2=2 1 (1*2)+2=4 2 (2*2)+2=6 3 (3*2)+2=8	
D. Verify that the <i>options</i> , when specified, are used to set stack size and priority properly. (OE0445, OE0448)				
6. Locate where the <i>options</i> parameters are being validated.	<p><b>Pass:</b> <i>Options</i> parameter validation is found (OE0445, OE0448)</p> <p><b>Fail:</b> A validation process does not exists for the <i>options</i> parameter (OE0445, OE0448)</p>			

OE_TC_227				
Steps	Expected Results	Actual Results	Comments	Test Result
7. Verify that when an <i>options</i> parameter has an id of STACK_SIZE_ID, then that <i>options</i> ' value is used to set the operating system's process/thread stack size.	<p><b>Pass:</b> When an <i>options</i> parameter has an id equal to STACK_SIZE_ID, then that <i>options</i>' value is used to set the operating system's process/thread stack size. (OE0445, OE0448)</p> <p><b>Fail:</b> When an <i>options</i> parameter has an id equal to STACK_SIZE_ID, then that <i>options</i>' value is not used to set the operating system's process/thread stack size. (OE0445, OE0448)</p>			
8. Verify that when an <i>options</i> parameter has an id of PRIORITY_ID, then that <i>options</i> ' value is used to set the operating system's process/thread priority.	<p><b>Pass:</b> When an <i>options</i> parameter has an id equal to PRIORITY_ID, then that option's value is used to set the operating system's process/thread priority. (OE0445, OE0448)</p> <p><b>Fail:</b> When an <i>options</i> parameter has an id equal to PRIORITY_ID, then that option's value is not used to set the operating system's process/thread priority. (OE0445, OE0448)</p>			
<b>E. Verify that the ExecutableDevice executes the function or file identified by the input <i>name</i> parameter. (OE0445)</b>				

OE_TC_227				
Steps	Expected Results	Actual Results	Comments	Test Result
9. Verify that the ExecutableDevice executes the function or file identified by the input <i>name</i> parameter.	<b>Pass:</b> The ExecutableDevice executes the function or file identified by the input <i>name</i> parameter. (OE0445)  <b>Fail:</b> The ExecutableDevice does not execute the function or file identified by the input <i>name</i> parameter. (OE0445)			
<b>End of Test</b>				

Test Recording Log – OE_TC_227					
Step1 (Locate ExecutableDevice interface )	Step2 & 3 (parameters converted to argv() type)	Step4 & 5 ( <i>parameters</i> are managed as id/value pairs)	Step7 (stack size <i>options</i> properly processed)	Step8 (priority <i>options</i> properly processed)	Step9 ( <i>named</i> file or function is executed)



## Test Summary OE\_TC\_227

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0445\_\_\_\_\_

OE0446\_\_\_\_\_

OE0447\_\_\_\_\_

OE0448\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.39. OE\_TC\_229 - Device :: allocateCapacity raises exceptions****Test Case Number:** OE\_TC\_229

Device::allocateCapacity raises exceptions

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0409	The <i>allocateCapacity</i> operation shall raise the InvalidCapacity exception, when the input capacities parameter contains invalid properties or when attributes of those CF Properties contain an unknown id or a value of the wrong data type.
OE0410	The <i>allocateCapacity</i> operation shall raise the InvalidState exception, when the Device's adminState is not UNLOCKED or operationalState is DISABLED.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-62 through 3-63, Section 3.1.3.3.1.5.1.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-2, C-24 thru C-25, C-27

**Test Objective**

This test case verifies OE0409 and OE0410. The objective of this test case is to verify that the *allocateCapacity* operation shall raise the InvalidCapacity exception, when the input capacities parameter contains invalid properties or when attributes of those CF::Properties contain an unknown id or a value of the wrong data type. Furthermore, the objective of this test case is to verify that the *allocateCapacity* operation shall raise the InvalidState exception, when the Device's adminState is not UNLOCKED or operationalState is DISABLED.

**Places to Verify**

All implementations of the Device interface for the operating environment under test.

**IDL References****Data**

```
typedef sequence <DataType> Properties;  
struct DataType {string id; any value;};
```

### Exceptions

```
exception InvalidCapacity { string msg; CF::Properties capacities; };  
exception InvalidState { string msg; };
```

### Operations

boolean allocateCapacity (in CF::Properties capacities) raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);

### Preconditions

- The Device interface source code files are available.

### Test Description

For each implementation of the Device interface, locate the code that implements the *allocateCapacity* operation for a device and perform the following:

- A. Verify the *allocateCapacity* operation shall raise the InvalidCapacity exception, when the input capacities parameter contains invalid properties or when attributes of those CF::Properties contain an unknown id or a value of the wrong data type. (OE0409)
  1. **Pass:** The InvalidCapacity exception is raised, when the input capacities parameter contains invalid properties or when attributes of those CF::Properties contain an unknown id or a value of the wrong data type.
  2. **N/A:** There are no implementations of the Device interface *allocateCapacity* operation.
  3. **Fail:** The InvalidCapacity exception is not raised when one or more input capacity parameter is invalid.
  4. **Fail:** The InvalidCapacity exception is not raised when an id is not known.
  5. **Fail:** The InvalidCapacity exception is not raised when a value is not the correct type.
- B. Verify that the InvalidState exception is raised when the device's adminState is not UNLOCKED or the operationalState is DISABLED. (OE0410)
  1. **Pass:** The InvalidState exception is raised when the device's adminState is not UNLOCKED or the operationalState is DISABLED.
  2. **Fail:** The InvalidState exception is not raised when the adminState is not UNLOCKED.
  3. **Fail:** The InvalidState exception is not raised when the operationalState is DISABLED.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_229				
Steps	Expected Results	Actual Results	Comments	Test Result
For each implementation of the Device interface do the following:				
A. Verify the <i>allocateCapacity</i> operation shall raise the <i>InvalidCapacity</i> exception, when the input capacities parameter contain invalid property or when attributes of those CF::Properties contain an unknown id or a value of the wrong data type. (OE0409)				
1. Locate the implementation of the <i>allocateCapacity</i> operation and record the implementation file name.	<b>N/A:</b> There are no implementations of the Device interface <i>allocateCapacity</i> operation. (OE0409)			
2. Confirm that the <i>allocateCapacity</i> operation shall raise the <i>InvalidCapacity</i> exception when the input capacities parameter contains invalid property.	<b>Pass:</b> The <i>InvalidCapacity</i> exception is raised when one or more input capacity parameter is invalid. (OE0409)  <b>Fail:</b> The <i>InvalidCapacity</i> exception is not raised when one or more input capacity parameter is invalid. (OE0409)		The <i>allocateCapacity</i> operation will contain code to validate the parameters. If an invalid property is contained it should raise an <i>InvalidCapacity</i> exception.	
3. Confirm that the <i>allocateCapacity</i> operation shall raise the <i>InvalidCapacity</i> exception when attributes of those CF::Properties contain an unknown id or a value of the wrong data type.	<b>Pass:</b> The <i>InvalidCapacity</i> exception is raised when an id is not known or a value of the wrong data type. (OE0409)  <b>Fail:</b> The <i>InvalidCapacity</i> exception is not raised when an id is not known. (OE0409)  <b>Fail:</b> The <i>InvalidCapacity</i> exception is not raised when a value is not the correct type. (OE0409)		The <i>allocateCapacity</i> operation will contain code to validate the parameters. If an unknown id or a value of the wrong type is contained it should raise an <i>InvalidCapacity</i> exception.	

OE_TC_229				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>B. Verify that the InvalidState exception is raised when the device's adminState is not UNLOCKED or the operationalState is DISABLED. (OE0410)</b>				
4. Verify the InvalidState exception is raised when the device's adminState is not UNLOCKED or the operationalState is DISABLED.	<b>Pass:</b> The InvalidState exception is raised when the device's adminState is not UNLOCKED or the operationalState is DISABLED. (OE0410)  <b>Fail:</b> The InvalidState exception is not raised when the adminState is not UNLOCKED. (OE0410)  <b>Fail:</b> The InvalidState exception is not raised when the operationalState is DISABLED. (OE0410)			
<b>End of Test</b>				

Test Recording Log – OE_TC_229			
Step1 (file name)	Steps 2-3 (InvalidCapacity exception)	Step4 (InvalidState exception)	Notes

## Test Summary OE\_TC\_229

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected  
**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.  
**Untested:** Condition which is not testable  
**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0409 \_\_\_\_\_

OE0410 \_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

**B.3.40. OE\_TC\_230 - Device :: deallocateCapacity raises InvalidState****Test Case Number:** OE\_TC\_230

Device::deallocateCapacity raises InvalidState

**Requirements**

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0417	The deallocateCapacity operation shall raise the InvalidState exception, when the device's adminState is LOCKED or operationalState is DISABLED.

**References**

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-64, Section 3.1.3.3.1.5.2.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-2, C-24, C-27

**Test Objective**

This test case verifies OE0417. The objective of this test case is to verify that the deallocateCapacity operation raises the InvalidState exception, when the device's adminState is LOCKED or operationalState is DISABLED.

**Places to Verify**

All implementations of the Device interface operation deallocateCapacity.

**IDL References****Data**

typedef sequence <DataType> Properties;

**Exceptions**

exception InvalidState { string msg; };

**Operations**

void deallocateCapacity (in CF::Properties capacities) raises (CF::Device::InvalidCapacity, CF::Device::InvalidState);



## Preconditions

- The Device interface source code files are available.

## Test Description

- A. Locate all implementations of the Device interface. (OE0417)
  1. **Fail:** There are no implementations of the Device interface.
- B. For each implementation of the Device interface operation deallocateCapacity, verify that the InvalidState exception is raised when the device's adminState is LOCKED or the operationalState is DISABLED. (OE0417)
  - a. **Pass:** The InvalidState exception is raised when the device's adminState is LOCKED or the operationalState is DISABLED.
  - b. **Fail:** The InvalidState exception is not raised when the adminState is LOCKED.
  - c. **Fail:** The InvalidState exception is not raised when the operationalState is DISABLED.

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.

2. The Test Recording Log is intended to record data for each step that requires recording of data.

OE_TC_230				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all implementations of the Device interface. (OE0417)</b>				
1. Record the file names for all of the files that implement the Device deallocateCapacity operation.	<b>Fail:</b> There are no implementations of the Device interface.			
<b>B. For each implementation of the Device interface operation deallocateCapacity, verify that the InvalidState exception is raised when the device's adminState is LOCKED or the operationalState is DISABLED. (OE0417)</b>				
2. Verify that the InvalidState exception is raised, by the deallocate operation, when the device's adminState is LOCKED.	<b>Pass:</b> The InvalidState exception is raised when the device's adminState is LOCKED.  <b>Fail:</b> The InvalidState exception is not raised when the device's adminState is LOCKED.			
3. Verify that the InvalidState exception is raised, by the deallocate operation, when the device's operationalState is DISABLED.	<b>Pass:</b> The InvalidState exception is raised when the operationalState is DISABLED.  <b>Fail:</b> The InvalidState exception is not raised when the operationalState is DISABLED			
<b>End of Test</b>				

Test Recording Log – OE_TC_230		
Step1 (implementation files)	Step2 (adminState)	Step3 (operationalState)

## Test Summary OE\_TC\_230

Once testing is complete for every component of the OE under test, report the test result as follows:

**Pass:** No failures detected

**Fail:** Failure(s) detected in Step(s)(x). Failure of any associated criteria results in a failure of a requirement.

**Untested:** Condition which is not testable

**N/A:** Not Applicable

### Overall Test Result (Pass, Fail, Untested, or N/A):

OE0417\_\_\_\_\_

### Failed Items (Section/Step Number):

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

**Test Engineer:** \_\_\_\_\_

**Date Tested:** \_\_\_\_\_

**Witness:** \_\_\_\_\_

### Index of Test Case Titles for Manual Tests

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
AEP Applications have no abnormal termination	5	31	B.5.4.	OE_TC_021
AEP file mode creation masks	5	37	B.5.5.	OE_TC_044
AEP mandatory functions	5	142	B.5.18.	OE_TC_131
AggregateDevice	3	191	B.3.25.	OE_TC_096
AggregateDevice devices	3	12	B.3.2.	OE_TC_022
AggregateDevice :: addDevice	3	20	B.3.3.	OE_TC_023
AggregateDevice :: addDevice FAILURE_ALARM	3	29	B.3.4.	OE_TC_025
AggregateDevice :: addDevice raises InvalidObjectReference	3	35	B.3.5.	OE_TC_027
AggregateDevice :: removeDevice	3	42	B.3.6.	OE_TC_028
AggregateDevice :: removeDevice FAILURE_ALARM	3	48	B.3.7.	OE_TC_029
AggregateDevice :: removeDevice raises InvalidObjectReference	3	54	B.3.8.	OE_TC_030
Application :: releaseObject releases all objects	2	96	B.2.16.	OE_TC_108
Application Attributes	2	258	B.2.39.	OE_TC_233
Application Delegates Implementation of Resource operations	2	237	B.2.38.	OE_TC_189
ApplicationFactory :: create	2	84	B.2.14.	OE_TC_074
ApplicationFactory :: create deallocates capacity on devices	2	201	B.2.32.	OE_TC_162
ApplicationFactory :: create defines an order for initialization	2	212	B.2.34.	OE_TC_165
ApplicationFactory :: create does property comparisons	2	119	B.2.20.	OE_TC_121
ApplicationFactory :: create establishes connections for named applications	2	207	B.2.33.	OE_TC_164
ApplicationFactory :: create raises CreateApplicationError	2	138	B.2.23.	OE_TC_125
ApplicationFactory Attributes	2	258	B.2.41.	OE_TC_236
Base Application Interfaces	1	263	B.1.35.	OE_TC_102
Base Device Interfaces	3	71	B.3.10.	OE_TC_062
ComponentIdentifier's execute parameter	5	200	B.5.22.	OE_TC_161
CORBA :: CosEventComm	5	52	B.5.8.	OE_TC_063
CORBA :: Log Producer	5	15	B.5.2.	OE_TC_002

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
CORBA :: NamingService	5	5	B.5.1.	OE_TC_001
Device :: adminState attribute changes	3	154	B.3.20.	OE_TC_088
Device :: adminState commanded to be LOCKED	3	91	B.3.12.	OE_TC_080
Device :: allocateCapacity	3	6	B.3.1.	OE_TC_004
Device :: allocateCapacity	3	105	B.3.13.	OE_TC_081
Device :: allocateCapacity BUSY	3	114	B.3.14.	OE_TC_082
Device :: allocateCapacity Failure	3	122	B.3.15.	OE_TC_083
Device :: allocateCapacity raises exceptions	3	299	B.3.39.	OE_TC_229
Device :: allocateCapacity's acceptable properties	3	217	B.3.28.	OE_TC_118
Device :: deallocateCapacity	3	128	B.3.16.	OE_TC_084
Device :: deallocateCapacity raises InvalidCapacity	3	141	B.3.18.	OE_TC_086
Device :: deallocateCapacity raises InvalidState	3	305	B.3.40.	OE_TC_230
Device :: deallocateCapacity usageState	3	134	B.3.17.	OE_TC_085
Device :: Logical Device - CORBA	3	177	B.3.23.	OE_TC_094
Device :: Logical Device - CORBA Register	3	184	B.3.24.	OE_TC_095
Device :: Logical Device allocation properties	3	206	B.3.27.	OE_TC_098
Device :: Logical Device executable parameters	3	168	B.3.22.	OE_TC_092
Device :: Logical Devices - CF Interfaces	3	198	B.3.26.	OE_TC_097
Device :: releaseObject	3	147	B.3.19.	OE_TC_087
Device :: releaseObject raises ReleaseError exception	3	162	B.3.21.	OE_TC_089
Device :: usageState	3	77	B.3.11.	OE_TC_079
Device Attributes	3	261	B.3.35.	OE_TC_218
Device operationalState	3	60	B.3.9.	OE_TC_058
DeviceManager :: getComponentImplementationId	2	89	B.2.15.	OE_TC_090
DeviceManager Attributes	2	247	B.2.40.	OE_TC_224
DeviceManager Register	2	17	B.2.3.	OE_TC_033
DeviceManager Register and the DCD file	2	23	B.2.4.	OE_TC_034

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
DeviceManager startup process	2	277	B.2.42.	OE_TC_285
DeviceManager's execparamproperties	2	101	B.2.17.	OE_TC_112
Domain Manager logs defined in DMD file	2	49	B.2.8.	OE_TC_054
Domain Manager services defined in DMD file	2	61	B.2.10.	OE_TC_056
Domain Profile	5	58	B.5.9.	OE_TC_070
Domain Profile files	5	114	B.5.17.	OE_TC_119
DomainManager :: installApplication raises ApplicationAlreadyInstalled	2	11	B.2.2.	OE_TC_026
DomainManager :: installApplication raises ApplicationInstallationError	2	67	B.2.11.	OE_TC_057
DomainManager :: installApplication raises InvalidFileName	2	172	B.2.28.	OE_TC_132
DomainManager :: registerDevice	2	107	B.2.18.	OE_TC_113
DomainManager :: registerDevice raises RegisterError	2	132	B.2.22.	OE_TC_124
DomainManager :: registerDevice registers if device doesn't exist	2	230	B.2.37.	OE_TC_168
DomainManager :: registerDevice returns without error if device exists	2	224	B.2.36.	OE_TC_167
DomainManager :: registerDevice verifies input parameters	2	218	B.2.35.	OE_TC_166
DomainManager :: registerDeviceManager	2	6	B.2.1.	OE_TC_024
DomainManager :: registerDeviceManager establishes connections	2	188	B.2.30.	OE_TC_157
DomainManager :: registerDeviceManager raises RegisterError	2	146	B.2.24.	OE_TC_126
DomainManager :: registerDeviceManager sends event to ODM	2	180	B.2.29.	OE_TC_154
DomainManager :: registerService	2	73	B.2.12.	OE_TC_065
DomainManager :: registerService	2	79	B.2.13.	OE_TC_073
DomainManager :: registerService raises RegisterError	2	153	B.2.25.	OE_TC_127
DomainManager :: uninstallApplication raises ApplicationUninstallationError	2	125	B.2.21.	OE_TC_122
DomainManager :: unregisterDevice raises UnregisterError	2	166	B.2.27.	OE_TC_129
DomainManager :: unregisterDeviceManager	2	43	B.2.7.	OE_TC_053
DomainManager :: unregisterDeviceManager	2	194	B.2.31.	OE_TC_158
DomainManager :: unregisterDeviceManager raises UnregisterError	2	159	B.2.26.	OE_TC_128
DomainManager :: unregisterService	2	32	B.2.5.	OE_TC_051

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
DomainManager :: unregisterService client-side	2	38	B.2.6.	OE_TC_052
DomainManager :: unregisterService raises UnregisterError	2	55	B.2.9.	OE_TC_055
DTD files	5	108	B.5.16.	OE_TC_117
Event Service	5	23	B.5.3.	OE_TC_003
Exceptions from the Mounted FileSystem	4	18	B.4.2.	OE_TC_040
ExecutableDevice :: execute	3	290	B.3.38.	OE_TC_227
ExecutableDevice :: execute raises exceptions	3	278	B.3.37.	OE_TC_222
ExecutableDevice :: execute raises InvalidFileName	3	244	B.3.32.	OE_TC_135
ExecutableDevice :: terminate raises InvalidProcess	3	250	B.3.33.	OE_TC_145
ExecutableDevice Types	3	272	B.3.36.	OE_TC_221
File :: close	4	55	B.4.5.	OE_TC_067
File :: close raises FileException	4	60	B.4.6.	OE_TC_068
File :: read raises IOException	4	49	B.4.4.	OE_TC_066
File :: setFilePointer raises FileException	4	66	B.4.7.	OE_TC_069
File :: sizeOf raises FileException	4	215	B.4.29.	OE_TC_147
File :: write raises IOException	4	208	B.4.28.	OE_TC_146
File Attributes	4	262	B.4.36.	OE_TC_265
FileManager :: list	4	6	B.4.1.	OE_TC_031
FileManager :: mount	4	278	B.4.38.	OE_TC_276
FileManager :: mount raises InvalidFileName	4	194	B.4.26.	OE_TC_143
FileSystem :: copy	4	99	B.4.12.	OE_TC_106
FileSystem :: copy raises InvalidFileName when inputs are invalid	4	148	B.4.20.	OE_TC_137
FileSystem :: copy raises InvalidFileName when inputs are the same	4	202	B.4.27.	OE_TC_144
FileSystem :: create	4	106	B.4.13.	OE_TC_107
FileSystem :: create raises FileException	4	227	B.4.31.	OE_TC_149
FileSystem :: create raises InvalidFileName	4	165	B.4.22.	OE_TC_139
FileSystem :: exists	4	159	B.4.21.	OE_TC_138



Test Case Title	App B Volume #	Page	Section Number	Test Case Number
FileSystem:: exists raises InvalidFileName	4	254	B.4.35.	OE_TC_159
FileSystem:: list raises FileException	4	87	B.4.10.	OE_TC_104
FileSystem:: list raises InvalidFileName	4	78	B.4.9.	OE_TC_103
FileSystem:: mkdir	4	129	B.4.17.	OE_TC_114
FileSystem:: mkdir raises FileException	4	241	B.4.33.	OE_TC_151
FileSystem:: mkdir raises InvalidFileName	4	179	B.4.24.	OE_TC_141
FileSystem:: open raises FileException	4	233	B.4.32.	OE_TC_150
FileSystem:: open raises InvalidFileName	4	172	B.4.23.	OE_TC_140
FileSystem:: query Input Parameter	4	269	B.4.37.	OE_TC_275
FileSystem:: remove raises FileException	4	93	B.4.11.	OE_TC_105
FileSystem:: remove raises InvalidFileName	4	141	B.4.19.	OE_TC_136
FileSystem:: rmdir	4	135	B.4.18.	OE_TC_115
FileSystem:: rmdir raises FileException	4	247	B.4.34.	OE_TC_152
FileSystem:: rmdir raises InvalidFileName	4	187	B.4.25.	OE_TC_142
FileSystemcreate Responsibilities	4	286	B.4.39.	OE_TC_283
FileSystemFilename Lengths	4	38	B.4.3.	OE_TC_060
FileSystem::copy raises FileException	4	221	B.4.30.	OE_TC_148
FileSystem's CREATED_TIME_ID property	4	111	B.4.14.	OE_TC_109
FileSystem's LAST_ACCESS_TIME_ID property	4	123	B.4.16.	OE_TC_111
FileSystem's MODIFIED_TIME_ID property	4	117	B.4.15.	OE_TC_110
Framework Control Interfaces	2	112	B.2.19.	OE_TC_116
Framework Services Interfaces	4	72	B.4.8.	OE_TC_091
General Rules : Higher Order Language	5	90	B.5.13.	OE_TC_099
Hardware Critical Interfaces	5	103	B.5.15.	OE_TC_101
Legacy Software interfaces	5	95	B.5.14.	OE_TC_100
LifeCycle :: initialize raises InitializeError	1	168	B.1.19.	OE_TC_037
LifeCycle :: releaseObject	1	173	B.1.20.	OE_TC_038

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
LifeCycle :: releaseObject raises ReleaseError	1	179	B.1.21.	OE_TC_039
LoadableDevice :: load	3	256	B.3.34.	OE_TC_153
LoadableDevice :: load raises InvalidFileName	3	230	B.3.30.	OE_TC_133
LoadableDevice :: load raises LoadFail	3	224	B.3.29.	OE_TC_130
LoadableDevice :: unload raises InvalidFileName	3	237	B.3.31.	OE_TC_134
Mandatory interfaces of the AEP	5	43	B.5.6.	OE_TC_059
Minimum CORBA	5	47	B.5.7.	OE_TC_061
Name Binding's execute parameter	5	194	B.5.21.	OE_TC_160
Naming Context creation	5	186	B.5.20.	OE_TC_156
Naming Context's execute parameter	5	180	B.5.19.	OE_TC_155
Networking AEP	5	208	B.5.23.	OE_TC_290
OMGLightweightLogService :: administrative_state	1	37	B.1.5.	OE_TC_009
OMGLightweightLogService :: clear_log	1	131	B.1.14.	OE_TC_018
OMGLightweightLogService :: destroy	1	139	B.1.15.	OE_TC_019
OMGLightweightLogService :: get_availability_status	1	31	B.1.4.	OE_TC_008
OMGLightweightLogService :: get_n_records	1	20	B.1.2.	OE_TC_006
OMGLightweightLogService :: get_operational_state	1	45	B.1.6.	OE_TC_010
OMGLightweightLogService :: get_record_id_from_time	1	51	B.1.7.	OE_TC_011
OMGLightweightLogService :: LogFullAction	1	25	B.1.3.	OE_TC_007
OMGLightweightLogService :: LogStatus	1	8	B.1.1.	OE_TC_005
OMGLightweightLogService :: retrieve_records	1	56	B.1.8.	OE_TC_012
OMGLightweightLogService :: retrieve_records_by_level	1	67	B.1.9.	OE_TC_013
OMGLightweightLogService :: retrieve_records_by_producer_id	1	78	B.1.10.	OE_TC_014
OMGLightweightLogService :: retrieve_records_by_producer_name	1	91	B.1.11.	OE_TC_015
OMGLightweightLogService :: write_record	1	117	B.1.13.	OE_TC_017
OMGLightweightLogService :: write_records	1	103	B.1.12.	OE_TC_016
Port :: connectPort	1	147	B.1.16.	OE_TC_020

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
Port :: connectPort raises OccupiedPort	1	155	B.1.17.	OE_TC_032
Port :: disconnectPort	1	161	B.1.18.	OE_TC_035
PropertySet :: configure	1	212	B.1.27.	OE_TC_047
PropertySet :: configure property minimums	1	218	B.1.28.	OE_TC_048
PropertySet :: configure raises PartialConfiguration	1	225	B.1.29.	OE_TC_049
Provided interfaces described as provides ports	5	67	B.5.10.	OE_TC_075
Required interfaces described as uses ports	5	73	B.5.11.	OE_TC_076
Resource :: start raises StartError	1	231	B.1.30.	OE_TC_050
Resource :: stop	1	241	B.1.32.	OE_TC_071
Resource :: stop raises StopError	1	236	B.1.31.	OE_TC_064
SCA APIs and non-IDL interfaces	5	79	B.5.12.	OE_TC_077
TestableObject :: runTest exception parameter	1	257	B.1.34.	OE_TC_093
TestableObject :: runTest parameters	1	246	B.1.33.	OE_TC_072
TestableObject :: runTest returns results	1	195	B.1.24.	OE_TC_043
TestableObject :: runTest uses testId parameter	1	184	B.1.22.	OE_TC_041
TestableObject :: runTest uses testValues parameter	1	189	B.1.23.	OE_TC_042
TestableObject :: runTest validates parameters	1	206	B.1.26.	OE_TC_046
TestableObject :: runTest validates test Values parameter	1	201	B.1.25.	OE_TC_045
<b>End of Table</b>				