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 2
Test Procedures for Framework Control 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

APPENDIX B VOLUME 2 TEST PROCEDURES

B.2. Volume 2 Framework Control Interfaces

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

B.2.1. OE_TC_024 - DomainManager :: registerDeviceManager

Test Case Number: OE_TC_024

DomainManager:: registerDeviceManager()

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0732	The *registerDeviceManager* operation shall raise the CF InvalidProfile exception when the device manager's DCD file and the DCD's referenced files do not exist.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-32 thru 3-50, Section 3.1.3.2.3
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-48, Section D.7.

Test Objective

This test case verifies OE0732. The objective of this test is to verify that an InvalidProfile exception is raised by the DomainManager when a DeviceManager's DCD file or any file referenced within the DCD file does not exist.

Places to Verify

Devices, DeviceManager

IDL References

Operations

void registerDeviceManager (in CF::DeviceManager deviceMgr)

raises (CF::InvalidObjectReference, CF::InvalidProfile, CF::DomainManager::RegisterError);

Preconditions

- All OE source files are available.

Test Description

A. Verify that the DomainManager's *registerDeviceManager* operation throws the InvalidProfile exception if it cannot find the DCD file. (OE0732)
   1. **Pass:** The InvalidProfile exception is thrown.
   2. **Fail:** The InvalidProfile exception is not thrown.

B. Verify that the DomainManager's *registerDeviceManager* operation throws the InvalidProfile exception if it cannot find a file referenced by the DCD file. (OE0732)
   1. **Pass:** The InvalidProfile exception is thrown.
   2. **Fail:** The InvalidProfile exception is not thrown.

Semi-automated Test Steps

After running JTAP's DomainManager registerDeviceManager InvalidProfile test, perform manual steps 1 through 3.

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_024				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the DomainManager's *registerDeviceManager* operation throws the InvalidProfile exception if it cannot find the DCD file. (OE0732)				
1. Locate the code for the DomainManager and its *registerDeviceManager* operation Record the name of the file.	The code is found.		File containing the operation may look like the following: void registerDeviceManager (in DeviceManager deviceMgr) raises (InvalidObjectReference, InvalidProfile, registerError);	
2. In the *registerDeviceManager* code locate where the *.dcd.xml is NOT found. Verify that the code raises the InvalidProfile exception.	**Pass**: The InvalidProfile exception is raised. (OE0732) **Fail**: The exception is NOT raised. (OE0732)			
B. Verify that the DomainManager's *registerDeviceManager* operation throws the InvalidProfile exception if it cannot find a file referenced by the DCD file. (OE0732)				
3. Locate the *registerDeviceManager* code where a file referenced by the DCD file is NOT found. Verify that the code raises the InvalidProfile exception.	**Pass**: The InvalidProfile exception is raised. (OE0732) **Fail**: The exception is NOT raised. (OE0732)			
End of Test				

Test Recording Log – OE_TC_024		
Step 1 (source code file)	Step 2 (Exception raised?)	Step 3 (Exception raised?)

Test Summary - OE_TC_024

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):

OE0732 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.2. OE_TC_026 - DomainManager :: installApplication raises ApplicationAlreadyInstalled

Test Case Number: OE_TC_026

Domain Manager:: installApplication()

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0722	The *installApplication* operation shall raise the ApplicationAlreadyInstalled exception when the softwareassembly element id attribute of the referenced application is the same as a previously registered application.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-43, Section 3.1.3.2.3.6.3.5 Page 3-34, Section 3.1.3.2.3.3.13

Test Objective

This test case verifies OE0722. The objective of this test is to verify that application installations are properly tracked by the DomainManager, that an attempt to install an already installed application is properly detected and that the appropriate exception is raised.

Places to Verify

Domain Manager

IDL References

Operations

```
void installApplication (in string profileFileName)
  raises (CF::InvalidProfile, CF::InvalidFileName, CF::DomainManager::ApplicationInstallationError,
          CF::DomainManager:: ApplicationAlreadyInstalled);
```

Preconditions

- The source code files for the DomainManager are available.

Test Description

A. Verify that the OE maintains a record of installed applications. (OE0722)
   1. **Untested**: No application can be installed.
   2. **Fail:** The OE does not keep a record of installed applications.

B. Verify that an attempt to install one of the applications already installed will be detected by the OE. Verify that this results in the ApplicationAlreadyInstalled exception being thrown. (OE0722)
   1. **Pass:** An attempt to install an already installed application results in the ApplicationAlreadyInstalled exception being thrown.
   2. **Fail:** An attempt to install an already installed application results in no exception being raised
   3. **Fail:** An attempt to install an already installed application results in some exception other than the ApplicationAlreadyInstalled exception being raised.

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_026				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the OE maintains a record of installed applications. (OE0722)				
1. Review the source code for the OE *installApplication()* operation.	The *installApplication* operation is implemented in the source code.			
2. Verify that the OE supports the installation of waveform applications.	**Untested**: No application can be installed. (OE0722)			
3. Verify that the code retains a list: *applicationFactories*, containing the Application Factories that have been installed.	Defined member attribute *applicationFactories*. **Fail**: The OE does not keep a record of installed applications. (OE0722)		The DomainManager stores the list of applications as a list of application factories, since the application factory actually creates the application.	
4. Verify that a successful *installApplication()* operation will add to the *applicationFactories* sequence.	A successful *installApplication()* operation will add the application / application factory to the *applicationFactories* sequence.			
B. Verify that an attempt to install one of the applications already installed will be detected by the OE. Verify that this results in the ApplicationAlreadyInstalled exception being thrown. (OE0722)				
5. Verify that the *installApplication()* operation iterates through the existing *applicationFactories* sequence.	**Pass:** The *applicationFactories* sequence, stored in the Domain Manager, is checked to determine if the name of an already installed application matches the input parameter. (OE0722) **Fail**: The *applicationFactories* sequence, stored in the Domain Manager, is not checked to determine if the name of an already installed application matches the input parameter. (OE0722)			

OE_TC_026				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Verify that the *installApplication()* operation will raise the *applicationAlreadyInstalled* exception if it finds a match among the existing *applicationFactories* sequence.	**Pass**: An attempt to install an already installed application results in the ApplicationAlreadyInstalled exception being thrown. **Fail**: An attempt to install an already installed application results in some other exception being raised (not the ApplicationAlreadyInstalled exception). (OE0722)			
End of Test				

Test Recording Log – OE_TC_026				
Step 1&2 (Source code found)	Step 3 (Application Factory list maintained)	Step 4 (Applications / Application Factories get added)	Step 5 (Iterates thru list)	Step 6 (Exception raised)

Test Summary OE_TC_026

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):

OE0722_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.3. OE_TC_033 - DeviceManager Register

Test Case Number: OE_TC_033

CF::DeviceManager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0473	The device manager upon start up shall register itself with a domain manager.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-52, Section 3.1.3.2.4.5
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-48-53, Section D.7

Test Objective

This test case verifies OE0473. The objective of this test is to verify that all device managers register with a domain manager when the device manager starts.

Places to Verify

DeviceManagers

IDL References

Operations

void registerDeviceManager (in DeviceManager deviceMgr) raises (InvalidObjectReference, InvalidProfile, RegisterError);

Preconditions

- The source code files are available.
- The Domain Profile files are available

Test Description

A. From the DCD files provided, determine the number of device managers and their names. (OE0473)
   1. **Untested**: No DCD files are provided.
B. For each device manager found in step A, perform the following:
   1. Locate the source code for the device manager (OE0473).
      a. **Pass:** The source code is located.
      b. **N/A:** The source code is not located.
   2. Verify that the *registerDeviceManager* operation of the domain manager is invoked during the device manager's startup (OE0473).
      a. **Pass:** The *registerDeviceManager* operation is invoked.
      b. **Fail:** The *registerDeviceManager* operation is not invoked.

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_033				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. From the DCD files provided, determine the number of device managers and their names. (OE0473)				
1. Locate the DCD file for each device manager and record the name of each device manager.	**Untested**: No DCD files are provided. (OE0473)		A device manager's id and name are defined in the DCD file as in the following example: <deviceconfiguration id="DCE:12345678-2222-4444-7777-098765432111" name="**SomeManager**"> … </deviceconfiguration>	
B. For each device manager found in step A, perform the following:				
1. Locate the source code for the device manager (OE0473).				
2. Locate the source code for the device manager and record the name of the file.	**Pass**: The source code is located. (OE0473) **Fail**: The source code is not located. (OE0473)		The development engineer, if available, is the best source for this information.	
2. Verify that the *registerDeviceManager* operation of the domain manager is invoked during the device manager's startup (OE0473).				
3. Locate the creation and initialization code for the device manager.	**Pass**: The startup code is located. (OE0473) **Fail**: The startup code is not located. (OE0473)			

OE_TC_033				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the creation or initialization code invokes the *registerDeviceManager* operation of the domain manager.	**Pass**: The *registerDeviceManager* operation is invoked properly. (OE0473) **Fail**: The *registerDeviceManager* operation is not invoked properly (OE0473)		If the device manager inherits from another class, this may be difficult to determine.	
5. Verify that the input parameter is the device manager object.	**Pass**: The input parameter is the device manager object. (OE0473) **Fail**: The input parameter is not the device manager object. (OE0473)			
End of Test				

Test Recording Log – OE_TC_033				
Step1 (device manager name)	**Step2** (Source code file)	**Step3** (startup code located)	**Step4** (*registerDeviceManager is invoked*)	**Step5** (parameter is the device manager reference)

Test Summary OE_TC_033

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):

OE0473_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.4. OE_TC_034 - DeviceManager Register and the DCD file

Test Case Number: OE_TC_034

CF::DeviceManager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0466	The readonly identifier attribute shall contain the instance-unique identifier for a Device Manager
OE0467	The identifier shall be identical to the *deviceconfiguration* element *id* attribute of the Device Manager's Device Configuration Descriptor (DCD) file.
OE0474	A Device Manager shall use the information in the Device Manager's DCD for determining: 1. Services to be deployed for this Device Manager (for example, log(s)), 2. Devices to be created for this Device Manager (when the DCD deployondevice element is not specified then the DCD componentinstantiation element is deployed on the same hardware device as the Device Manager), 3. Devices to be deployed on (executing on) another device, 4. Devices to be aggregated to another device, 5. Mount point names for FileSystems, 6. The DeviceManager's identifier attribute value which is the DCD's id attribute value, and 7. The DeviceManager's label attribute value, which is the DCD's name attribute value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2, FINAL / 15 May 2006	Page 3-51, Section 3.1.3.2.4.4.1; Page 3-52, Section 3.1.3.2.4.5
SCA Appendix D: Domain Profile	version 2.2.2, FINAL / 15 May 2006	Page D-48

Test Objective

This test case verifies OE0466, OE0467, and OE0474. The objective of this test is to verify that the Device Manager's identifier is unique and is retrieved from the Device Manager's DCD. It will also verify that the other information from the DCD is used correctly to determine Services, Devices, mount point, and attributes of a Device Manager. Note that an id attribute in the DCD is required to be a DCE UUID. This test will verify that also.

Notice that requirement OE0474 has seven pieces of information it lists as coming from the DCD. These pieces of information are numbered in the requirement. Any references to a specific piece of information will be followed by the requirement tag, a hyphen and its number within the requirement. However, this notation is to assist backtracking to the sub-paragraph of the requirement and nothing more. Any failure of any of these hyphenated requirement numbers is a failure of OE0474.

Places to Verify

DeviceManager

IDL References

None.

Preconditions

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

Test Description

A. Identify all the DCD files for the OE and record them.(OE0466, OE0467, OE0474)
   1. **Untested**: There are no DCD files.

For each DCD file, perform the following:

B. Record the id and name of the deviceconfiguration element. (OE0474-6, OE0474-7)
C. Verify that the id is a valid, unique DCE UUID. (OE0466)
   1. **Pass:** The id is a valid DCE UUID
   2. **Fail**: The id is not a valid DCE UUID.
D. Identify the source code for the Device Manager. (OE0466, OE0467, OE0474)
   1. **Pass:** Source code is provided.
   2. **Fail:** No source code is provided.
E. Verify in the source code that the identifier attribute is set to the id from the DCD (OE0467, OE0474-6).
   1. **Pass**: The identifier attribute is set to the id from the DCD.
   2. **Fail**: The identifier attribute is not set to the id from the DCD.
F. Verify in the source code that the label attribute is set to the name from the DCD. (OE0474-7)
   1. **Pass:** The label attribute is set to the name from the DCD.
   2. **Fail**: The label attribute is not set to the name from the DCD.
G. Verify in the source code that the Services to be deployed are extracted from the DCD. (OE0474-1)
   1. **Pass:** The Services to be deployed are extracted from the DCD.
   2. **Fail**: The Services to be deployed are not extracted from the DCD.
H. Verify in the source code that the Devices to be created are extracted from the DCD. (OE0474-2)

1. **Pass:** The Devices to be created are extracted from the DCD.
2. **Fail**: The Devices to be created are not extracted from the DCD.

I. Verify in the source code that the Devices to be deployed on another device are extracted from the DCD. (OE0474-3)
1. **Pass:** The Devices to be deployed on another device are extracted from the DCD.
2. **Fail**: The Devices to be deployed on another device are not extracted from the DCD.
3. **N/A**: There are no Devices deployed on another device.

J. Verify in the source code that the Devices to be aggregated on another device are extracted from the DCD. (OE0474-4)
1. **Pass:** The Devices to be aggregated on another device are extracted from the DCD.
2. **Fail**: The Devices to be aggregated on another device are not extracted from the DCD.
3. **N/A**: There are no aggregated Devices.

K. Verify in the source code that the mount point names for FileSystems are extracted from the DCD. (OE0474-5)
1. **Pass:** The mount point names for FileSystems are extracted from the DCD.
2. **Fail**: The mount point names for FileSystems are not extracted from the DCD.

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_034				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify all the DCD files for the OE and record them.(OE0466, OE0467, OE0474)				
1. Locate the DCD files and record their names.	The DCD files are found. **Untested**: There are no DCD files (i.e. there are no Device Managers) .(OE0466, OE0467, OE0474)			
For each DCD file perform the following:				
B. Record the id and name of the deviceconfiguration element. (OE0474-6, OE0474-7)				
2. Record the id and name of the deviceconfiguration element.	**Pass**: The id and name are present, (OE0474) **Fail**: The id is missing. (OE0474) **Fail**: The name is missing. (OE0474)		The id and name can be found as in this example: <deviceconfiguration id="**DCE:12345678-1234-1234-1234-123456789012**" name="someDeviceManager"> … </deviceconfiguration>	
C. Verify that the id is a valid, unique DCE UUID. (OE0466)				

OE_TC_034				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the id is a DCE UUID.	The id is a DCE UUID. **Pass**: The id is a UUID. (OE0466) **Fail**: the id is not a valid UUID. (OE0466)		The DCE UUID format starts with the characters "DCE:" and is followed by the printable form of the 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.	
4. If there is more than 1 Device Manager, verify that these ID's are unique.	**Pass**: The ids are unique. (OE0466) **Fail**: An id is duplicated. (OE0466)			
D. Identify the source code for the Device Manager. (OE0466, OE0467, OE0474)				
5. Locate the source code for the Device Manager	**Pass:** Device Manager source code is found. (OE0466, OE0467, OE0474) **Fail:** No Device Manager source code is found. (OE0466, OE0467, OE0474)			
6. Locate the creation and initialization code for the Device Manager.	**Pass:** Device Manager source code is provided. (OE0466, OE0467, OE0474) **Fail:** No Device Manager source code is provided. (OE0466, OE0467, OE0474)		Most of this testing will involve the startup of the Device Manager.	
E. Verify in the source code that the label attribute is set to the name from the DCD. (OE0474-7)				

OE_TC_034				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Locate the code to set the value of the identifier attribute and verify that the value is retrieved from the DCD.	**Pass:** The identifier attribute data is retrieved from the DCD. (OE0474) **Fail:** The identifier attribute data is not retrieved from the DCD. (OE0474)			
F. Verify in the source code that the label attribute is set to the name from the DCD. (OE0474-7)				
8. Locate the code to set the value of the label attribute and verify that the value is retrieved from the DCD.	**Pass:** The label attribute data is retrieved from the DCD. (OE0474) **Fail:** The label attribute data is not retrieved from the DCD. (OE0474)			
G. Verify in the source code that the Services to be deployed are extracted from the DCD. (OE0474-1)				
9. Locate the code that deploys Services and verify that the Services are retrieved from the DCD.	**Pass:** The Services are retrieved from the DCD. (OE0474) **Fail:** The Services are not retrieved from the DCD. (OE0474)			
H. Verify in the source code that the Devices to be created are extracted from the DCD. (OE0474-2)				
10. Locate the code to create Devices and verify that the Devices to be created retrieved from the DCD.	**Pass:** The Devices to be created are retrieved from the DCD. (OE0474) **Fail:** The Devices to be created are not retrieved from the DCD. (OE0474)			
I. Verify in the source code that the Devices to be deployed on another device are extracted from the DCD. (OE0474-3)				

OE_TC_034				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
11. Locate the code to deploy Devices on another device and verify that the Devices are retrieved from the DCD.	**Pass:** The Devices are retrieved from the DCD. (OE0474) **Fail:** The Devices are not retrieved from the DCD. (OE0474) **N/A**: No Devices are deployed on another device. (OE0474)			
J. Verify in the source code that the Devices to be aggregated on another device are extracted from the DCD. (OE0474-4)				
12. Locate the code to aggregate Devices and verify that the Devices are retrieved from the DCD.	**Pass:** The Devices are retrieved from the DCD. (OE0474) **Fail:** The Devices are not retrieved from the DCD. (OE0474) **N/A**: No Devices are aggregated. (OE0474)			
K. Verify in the source code that the mount point names for FileSystems are extracted from the DCD. (OE0474-5)				
13. Locate the code to set the value of the mount point names for FileSystems and verify that the value is retrieved from the DCD.	**Pass:** The mount point names for FileSystems are retrieved from the DCD. (OE0474) **Fail:** The mount point names for FileSystems are not retrieved from the DCD. (OE0474)			
End of Test				

Test Recording Log – OE_TC_034				
Test Step	1st Device Manager	2nd Device Manager	3rd Device Manager	4th Device Manager
Step 1 (DCD file names)				
Step 2 (id and name)				
Step 3 (UUID)				
Step 4 (unique ID's)				
Step 5 (source code)				
Step 6 (initialization code)				
Step 7 (identifier attribute)				
Step 8 (label attribute)				
Step 9 (Services retrieved)				
Step 10 (Devices retrieved)				
Step 11 (deployed Devices)				
Step 12 (aggregate Devices)				
Step 13 (mount points retrieved)				

Test Summary OE_TC_034

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):

OE0466_____

OE0467_____

OE0474_____

Failed Items (Section/Step Number):

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.5. OE_TC_051 - DomainManager :: unregisterService

Test Case Number: OE_TC_051

DomainManager::*unregisterService*

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0327	The unregisterService operation shall remove the unregisteringService entry specified by the input name parameter from the domain manager.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-48, Section 3.1.3.2.3.6.8
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-21

Test Objective

This test case verifies requirement OE0327. The objective of this test is to verify that the *unregisterService* operation removes the unregisteringService entry specified by the input name parameter from the DomainManager.

Places to Verify

DomainManager

IDL References

```
void unregisterService
        (in Object unregisteringService, in string name)
        raises (InvalidObjectReference, UnregisterError);
```

Preconditions

- The Domain Manager source code files are available.

Test Description

A. Locate the *unregisterService* operation DomainManager source code. (OE0327)
   1. **Pass:** The operation *unregisterService* is found.
   2. **Fail:** The operation *unregisterService* is not found.

B. Verify that the input *name* parameter identifies the *unregisteringService* entry. (OE0327)
   1. **Pass:** The *unregisteringService* entry is identified by the input *name* parameter.
   2. **Fail:** The *unregisteringService* entry is not identified by the input *name* parameter.

C. Verify that the *unregisteringService* entry is removed from the DomainManager. (OE0327)
   1. **Pass:** The object reference of the *unregisteringService* entry is removed from the domain manager.
   2. **Fail:** The object reference of the *unregisteringService* entry still exists in the domain manager.

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_051				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the *unregisterService* operation DomainManager source code. (OE0327)				
1. Locate the Domain Manager source code.	**Pass:** the Domain Manager source code is found. (OE0327) **Fail:** the Domain Manager source code is not found. (OE0327)			
2. Locate the unregisterService code within the Domain Manager's source code and record the file name.	**Pass:** the unregisterService source code is found. (OE0327) **Fail:** the unregisterService source code is not found. (OE0327)		The application software engineer can be of assistance in locating the source code.	
B. Verify that the input *name* parameter identifies the *unregisteringService* entry. (OE0327)				
3. Verify that the input *name* parameter identifies the *unregisteringService* entry.	**Pass:** The *unregisteringService* entry is identified by the input *name* parameter. (OE0327) **Fail:** The *unregisteringService* entry is not identified by the input *name* parameter. (OE0327)			
C. Verify that the *unregisteringService* entry is removed from the DomainManager. (OE0327)				

OE_TC_051				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the *unregisteringService* entry is removed from the Domain Manager.	**Pass:** The object reference of the *unregisteringService* entry is removed from the domain manager. (OE0327) **Fail:** The object reference of the *unregisteringService* entry still exists in the domain manager. (OE0327)			
End of Test				

Test Recording Log – OE_TC_051			
Step1 (source code found)	**Step2** (unregisterService source code file name)	**Step3** (input *name* parameter)	**Step4** (removal of object reference)

Test Summary OE_TC_051

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):

OE0327_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.2.6. OE_TC_052 - DomainManager :: unregisterService client-side

Test Case Number: OE_TC_52

Domain Manager Operations::unregisterService

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0328	The unregisterService operation shall release (client-side CORBA release) the unregisteringService from the domain manager.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-48, Section 3.1.3.2.3.6.8.3

Test Objective

This test case verifies OE0328. The objective of this is to verify that the unregisterService operation releases the service on the client-side from the DomainManager.

Places to Verify

Domain Manager

IDL References

Exceptions

```
exception InvalidObjectReference { string msg };
exception UnregisterError {
        CF::ErrorNumberType errorNumber;
        string msg; };
```

Operations

```
void unregisterService (
        in Object unregisteringService, in string name)
                raises (InvalidObjectReference, UnregisterError);
```

Preconditions

- The source code files for the domain manager are available.

Test Description

A. Verify that the DomainManager *unregisterService* operation releases the *unregisteringService*. (OE0328)
   1. **Pass**: The *unregisteringService* is released from the CORBA environment.
   2. **Fail**: The *unregisterService* operation does not release the *unregisteringService* 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_052				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the DomainManager *unregisterService* operation releases the *unregisteringService*. (OE0328)				
1. Locate the *unregisterService* operation in the domain manager source code. Record the source code filename in the test log.	**Pass**: Source code located. (OE0328) **Untested**: Source code not located. (OE0328)			
2. Verify that the DomainManager *unregisterService* operation either (1) sets the service storage _var to nil to release the reference or (2) manually releases the reference (CORBA::release) when it is not stored in a _var. Record the result in the test log.	**Pass**: the service storage _var is set to nil, (OE0328) OR **Pass:** the reference is manually released. (OE0328) **Fail**: The unregisterService operation does not release the *unregisteringService*. (OE0328)			
End of Test				

Test Recording Log – OE_TC_052		
Step 1 (Source code filename)	Step 2 (Sets _var to nil)	Step 2 (Manually release)

Test Summary - OE_TC_052

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)**:**

OE0328_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.2.7. OE_TC_053 - DomainManager :: unregisterDeviceManager

Test Case Number: OE_TC_053

DomainManager::*unregisterDeviceManager*

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0274	The *unregisterDeviceManager* operation shall release all device(s) and service(s) associated with the device manager that is being unregistered.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-43, Section 3.1.3.2.3.6.4.3

Test Objective

This test case verifies OE0274. The objective of this test is to verify that the *unregisterDeviceManager* operation releases all device(s) and service(s) associated with the device manager that is being unregistered. Even though devices and services are found in the Device Configuration Descriptor file, it is not necessary to identify them. Reviewing the source code is sufficient to verify that the devices and services are release from the associated device manager.

Places to Verify

DomainManager, DeviceManagers

IDL References

Operations

```
void unregisterDeviceManager (in CF::DeviceManager deviceMgr)
        raises (CF::InvalidObjectReference, CF::UnregisterError);
```

Exceptions

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

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

Preconditions

- All of the source code files of the Core Framework are available.

Test Description

A. Locate all implementations of the *unregisterDeviceManager* operation in the CF source code. (OE0274)
   1. **Pass**: An implementation of the *unregisterDeviceManager* operation is found.
   2. **Fail**: An implementation of the *unregisterDeviceManager* operation is not found.

B. Verify that the *unregisterDeviceManager* operation releases all *devices* from the device manager. (OE0274).
   1. **Pass**: The *unregisterDeviceManager* operation releases all *devices* associated with the device manager being unregistered.
   2. **Fail**: The *unregisterDeviceManager* operation does not release a *device* associated with the device manager being unregistered.

C. Verify that the *unregisterDeviceManager* operation releases all *services* from the *device manager* (OE0274).
   1. **Pass**: The *unregisterDeviceManager* operation releases all *services* associated with the device manager being unregistered.
   2. **Fail**: The *unregisterDeviceManager* operation does not release a *service* associated with the device manager being unregistered.

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_053				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all implementations of the *unregisterDeviceManager* operation in the CF source code. (OE0274)				
1. Identify and record the source code file(s) that implements the *unregisterDeviceManager* operation.	**Pass**: The source code implementing the *unregisterDeviceManager* operation is located. (OE0274) **Fail**: The source code is not located. (OE0274)		An Integrated Development Environment (IDE) is the best tool for this search. void unregisterDeviceManager (in DeviceManager deviceMgr) raises (InvalidObjectReference, UnregisterError);	
B. Verify that the *unregisterDeviceManager* operation releases all *devices* from the device manager. (OE0274).				
2. Verify that the unregisterDeviceManager operation locates the Device Manager identified in the input parameter to unregisterDeviceManager.				
3. Verify that the Device Manager's record of all the *device(s)* associated to the Device manager is located.				

OE_TC_053				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the *unregisterDeviceManager* operation releases all devices associated with the device manager that is being unregistered.	**Pass**: The *unregisterDeviceManager* operation releases all devices associated with the device manager being unregistered. (OE0274) **Fail**: The *unregisterDeviceManager* operation does not release all devices associated with the device manager that is being unregistered. (OE0274)		Each device manager contains complete knowledge of a set of devices.	
C. Verify that the *unregisterDeviceManager* operation releases all *services* from the *device manager* (OE0274).				
5. Verify that the Device Manager's record of all the *service(s)* associated with the Device manager is located.				
6. Verify that the *unregisterDeviceManager* operation releases all service(s) associated with the device manager.	**Pass**: The *unregisterDeviceManager* operation releases all service associated with the device manager being unregistered. (OE0274) **Fail**: The *unregisterDeviceManager* operation does not release all services associated with the device manager that is being unregistered. (OE0274)		Each device manager contains complete knowledge of a set of services.	
End of Test				

Test Recording Log – OE_TC_053		
Step1 (list of source code files)	**Step4** (releases all devices associated with the unregistering device manager – Y/N?)	**Step6** (releases all services associated with the unregistering device manager – Y/N?)

Test Summary OE_TC_053

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):

OE0274_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.8. OE_TC_054 - Domain Manager logs defined in DMD file

Test Case Number: OE_TC_054

Domain Manager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0217	The logs utilized by the DomainManager implementation shall be defined in the DMD.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-35 thru 3-36, Section 3.1.3.2.3.5
SCA Appendix D: Domain Profile	Revision 2.2.2	Page D-54 thru D-55, Section D.8

Test Objective

This test case verifies OE0217. The objective of this test is to verify that all logs used by the DomainManager are defined in the DMD.

Places to verify

DomainManager and DMD

IDL References

None.

Preconditions

- The domain profile files are available.
- The DomainManager source code files are available.

Test Description

A. Verify that the domain manager uses one or more logs. (OE0217)
   1. **N/A**: The domain manager does not use a log.

B. Verify that each log used by the domain manager is in the domain manager's DMD file. (OE0217)

1. **Pass:** Every log used by the domain manager is described in the DMD file.
2. **Fail:** One or more logs used by the domain manager are not described in the DMD file.

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_054				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the domain manager uses one or more logs. (OE0217)				
1. Locate the DMD file for the domain manager of the OE and record its name.	**Pass**: The DMD is found. (OE0217) **Fail**: No DMD exists. (OE0217)		As a DMD is required for a domain manager, and a domain manager is required for the OE, this step should never fail.	
2. Examine the DMD and record the log services that it uses.	**N/A**: no log entries are in the DMD. (OE0217)		The log service is identified by a type of "log". The entry should look like: <services> <service> <usesidentifier>LogService </usesidentifier> <findby> <domainfinder name="xxxxx" type="log"/> </findby> </service> … </services> The name is what we need to record. They may have multiple log services.	
B. Verify that each log used by the domain manager is in the domain manager's DMD file. (OE0217)				

OE_TC_054				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Locate the DomainManager's code that outputs log messages and identify the log service.	**Pass**: The log service can be identified. (OE0217) **Fail**: The log service cannot be identified. (OE0217)		The DomainManager has several requirements to output log records; therefore, there should be code to output records. The operations involved are: write_record write_records Note also that they may have their own log implementation that hides the OMG lightweight log from the developer's code.	
4. Locate the instantiation of the log service and verify that its name is the name found in step 2.	**Pass**: The log service is the log service specified in the DMD. (OE0217) **Fail**: The log service is not the log service specified in the DMD. (OE0217)			
End of Test				

Test Recording Log – OE_TC_054		
Step 1 (DMD file)		
Step 2 (Log services)	Step 3 (log service name)	Step 4 (Log service is from DMD)

Test Summary OE_TC_054

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):

OE0217_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.9. OE_TC_055 - DomainManager :: unregisterService raises UnregisterError

Test Case Number: OE_TC_055

DomainManager::unregisterService raises UnregisterError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0337	The *unregisterService* operation shall raise the *UnregisterError* exception when an internal error exists which causes an unsuccessful unregistration.
OE0202	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-48, Section 3.1.3.2.3.6.8.5 Page 3-33, Section 3.1.3.2.3.3.8 Page 3-33, Section 3.1.3.2.3.3.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	C-17 and C-18

Test Objective

This test case verifies OE0337 and OE0202. The objective of this test is to verify that DomainManager's *unregisterService* operation will raise an *UnregisterError* exception when an unsuccessful unregistration occurs. This *UnregisterError* exception provides information about the error, such as the error number, which is a CF ErrorNumberType value.

Places to Verify

DomainManager

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_EMSGSIZE, 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 UnregisterError
{CF::ErrorNumberType errorNumber; string msg;};

Operations

void unregisterService (
in Object unregisteringService,
in string name
)
raises (CF::InvalidObjectReference,
CF::DomainManager::UnregisterError);

Preconditions

- The source code files are available.

Test Description

A. Verify that the source code for the *unregisterService* operation and *UnregisterError* exception code are found. (OE0337)
   1. **Pass:** The source code is found.
   2. **Fail:** The source code is not found.
B. Verify that the *UnregisterError* exception will be raised if an error is caught. (OE0337)
   1. **Pass:** The *UnregisterError* exception is raised for an error condition.
   2. **Fail:** The *UnregisterError* exception is not raised for an error condition.
C. Verify the presence of an error number with a CF ErrorNumberType value for this exception. (OE0202)
   1. **Pass:** The *UnregisterError* exception provides an error number with a CF ErrorNumberType value for the error condition.
   2. **Fail:** The *UnregisterError* 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_055				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the source code for the *unregisterService* operation and *UnregisterError* exception code are found. (OE0337)				
1. For the DomainManager's *unregisterService* operation and the *UnregisterError* exception, locate the source code file(s) and record the directory path and source code file name(s).	**Pass:** The *unregisterService* operation and the *UnregisterError* exception source code are located and the locations are recorded in the Test Recording Log. (OE0337) **Fail**: The *unregisterService* operation and the *UnregisterError* exception source code are not located. End the test. (OE0337)		To locate the right code, the OE software engineer can be of assistance. Samples: *void unregisterService (in Object unregisteringService, in string name) raises (InvalidObjectReference, UnregisterError);* *exception UnregisterError {CF::ErrorNumberType errorNumber; string msg; };*	
B. Verify that the *UnregisterError* exception will be raised if an error is caught. (OE0337)				
2. Examine the source code files and verify that when an internal error is caught, the *unregisterService* operation will raise the *UnregisterError* exception.	**Pass**: When an internal error is caught, the *unregisterService* operation will raise the *UnregisterError* exception. (OE0337) **Fail**: When an internal error is caught, the *unregisterService* operation will not raise the *UnregisterError* exception. (OE0337)			
C. Verify the presence of an error number with a CF ErrorNumberType value for this exception. (OE0202)				

OE_TC_055				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0202) **Fail:** The exception does not include an error number (ErrorNumber) of ErrorNumberType. (OE0202)			
End of Test				

Test Recording Log – OE_TC_055			
Step 1 (file name location)	Step 2 (UnregisterError exception)	Step 3 (Correct type)	Step 4 (value/argument)

Test Summary OE_TC_055

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 the requirement.
Untested: Condition which is not testable
N/A: Not Applicable

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

OE0337 _____

OE0202 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.2.10. OE_TC_056 - Domain Manager services defined in DMD file

Test Case Number: OE_TC_056

DomainManager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0218	The domain manager shall begin to use a service specified in the DMD once the service is successfully registered with the domain manager via the *registerDeviceManager* or *registerService* operations.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-36, Section 3.1.3.2.3.5
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-54, Section D.8.1

Test Objective

This test case verifies requirement, OE0218. The objective of this test case is to verify that the domain manager will only allow a service listed in the DomainManager Configuration Descriptor(DMD) file to be made available for use after it has been successfully registered by the *registerDeviceManager* or *registerService* operations.

Places to Verify

DomainManager

IDL References

Exceptions

```
exception InvalidObjectReference { string msg; };
exception DeviceManagerNotRegistered {};
exception RegisterError { CF::ErrorNumberType errorNumber; string msg; };
exception InvalidProfile { };
```

Operations

```
void registerService ( in Object registeringService, in CF::DeviceManager registeredDeviceMgr, in string name )
        raises (CF::InvalidObjectReference,
```

CF::DomainManager::DeviceManagerNotRegistered,
CF::DomainManager::RegisterError);
void registerDeviceManager (in CF::DeviceManager deviceMgr)
raises(CF::InvalidObjectReference, CF::InvalidProfile, CF::DomainManager::RegisterError);

Preconditions

- The source code files of the Core Framework, including the domain manager, are available.

Test Description

A. Identify the services from the DomainManager Configuration Descriptor(DMD) xml file. (OE0218)
   1. **N/A**: There are no services listed in the DMD file.
B. Verify that the *registerService* and *registerDeviceManager* operations exist in the domain manager source code. (OE0218)
   1. **Untested**: The *registerService* operation does not exist in the domain manager source code.
   2. **Untested**: The *registerDeviceManager* operation does not exist in the domain manager source code.

For each service listed in the DMD, perform the following step(s):

C. Verify that the domain manager only allows a service, which is listed in the DMD to be available for use after it has been successfully, registered by either the *registerService* or *registerDeviceManager* operations. (OE0218)
   1. **Pass**: The domain manager only allows a service listed in the DMD to be used after it has been successfully registered by either the *registerService* or *registerDeviceManager* operations.
   2. **Fail**: The domain manager allows a service listed in the DMD to be used before it has been successfully registered by either the *registerService* or *registerDeviceManager* operations.
   3. **Fail**: The domain manager does not make the service available for use after the service has successfully registered with the domain manager.

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_056				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the services from the DomainManager Configuration Descriptor (DMD) xml file. (OE0218)				
1. Identify the services that the Domain Manager uses from the DMD file. List the services from the DMD file.	**N/A**: No services are listed in the DMD file. The test ends here. (OE0218)		<services> <service> <usesidentifier>xxx </usesidentifier> <findby> <domainfinder type="log" name="yyy"> </domainfinder> </findby> </service> </services>	
B. Verify that the *registerDeviceManager* and *registerService* operations exist in the domain manager source code. (OE0218)				
2. Verify in the domain manager source code that the *registerService* operation exists. List the source code files of the registerService operation.	**Untested**: The *registerService* operation does not exist in the domain manager source code. (OE0218)			
3. Verify in the domain manager source code that the *registerDeviceManager* operation exists. List the source code files of the *registerDeviceManager* operation.	**Untested**: The *registerDeviceManager* operation does not exist in the domain manager source code. (OE0218)			
For each service listed in the DMD, perform the following step(s):				
C. Verify that the domain manager only allows a service, which is listed in the DMD to be available for use after it has been successfully, registered by either the *registerService* or *registerDeviceManager* operations. (OE0218)				

OE_TC_056				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Identify the location in the domain manager source code where the *registerService* and *registerDeviceManager* operations perform the registration of services.	The *registerService* and *registerDeviceManager* operations are located in the domain manager source code.		This is not a pass/fail step, but usually a domain manager performs the *registerService* and/or *registerDeviceManager* operations before it checks that the DMD service is registered and enables it for use.	
5. Verify that logic exists in the domain manager source code, which checks that the service specified in the DMD has been registered by either the *registerService* or *registerDeviceManager* operations before the domain manager makes it available for use.	**Pass**: The domain manager only allows a service listed in the DMD to be used after it has been successfully registered by either the registerService or registerDeviceManager operations.(OE0218) **Fail:** The domain manager allows a service listed in the DMD to be used before it has been successfully registered by either the registerService or registerDeviceManager operations. (OE0218) **Fail**: The domain manager does not make the service available for use after the service has successfully registered with the domain manager. (OE0218)		It may be helpful to search for the words, domainfinder or findby, since the domain manager may use these elements of the Domain Profile to locate the DMD service in the registered services (a possible, but not the only way). It may also be helpful to search for the term, log, since in most Core Frameworks the log is listed as one of the services in the DMD file.	
End of Test.				

Test Recording Log – OE_TC_056				
Step 1 (List of services from the DMD file.)	Step 2 (File name of *registerService* operation or "Untested" if it doesn't exist)	Step 3 (File name of *registerDeviceManager* operation or "Untested" if it doesn't exist)	Step 4 (*registerService registerDeviceManager* ops are located in DM source code.) (Y/N?)	Step 5 (logic exists in DM source code to check that the DMD service is registered before being made available for use) (Pass/Fail?)

Test Summary OE_TC_056

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):

OE00218_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.11. OE_TC_057 - DomainManager :: installApplication raises ApplicationInstallationError

Test Case Number: OE_TC_057

DomainManager::installApplication raises ApplicationInstallationError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0200	The error number shall indicate a CF ErrorNumberType value.
OE0261	The installApplication operation shall, upon unsuccessful application installation, write a FAILURE_ALARM log record to a domain manager's log.
OE0268	The installApplication operation shall raise the ApplicationInstallationError exception when the installation of the application file(s) was not successfully completed.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-33, Section 3.1.3.2.3.3.1 Page 3-42 thru 43, Section 3.1.3.2.3.6.3

Test Objective

This test case verifies OE0200, OE0261 and OE0268. The objective of this test is to verify the error handling of the domain manager's *installApplication* operation when installation of the application's files does not succeed. Furthermore, verify the exception contains an error number of the type CF::ErrorNumberType.

Places to Verify

DomainManager

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_EMSGSIZE,
        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 };

Exception

exception ApplicationInstallationError { ErrorNumberType errorNumber; string msg; };

Operations

void installApplication (in string profileFileName)
raises (CF::InvalidProfile,
CF::InvalidFileName,
CF::DomainManager::ApplicationInstallationError,
CF::DomainManager::ApplicationAlreadyInstalled);

Preconditions

- The OE source code files are available.

Test Description

A. Locate the DomainManager. (OE0200, OE0261, OE0268)
   1. **Fail**: No DomainManager can be located.
B. Verify that the *installApplication* operation detects installation failures of an application's files.(OE0200, OE0261, OE0268)
   1. **Fail:** Installation failures are not detected.
C. Verify that a FAILURE_ALARM log record is written to the domain manager's log when the *installApplication* operation detects installation failures of an application's files. (OE0261).
   1. **Pass:** A FAILURE_ALARM record is written to the domain manager's log.
   2. **Fail:** No FAILURE_ALARM record is written to the domain manager's log.
D. Verify that the *ApplicationInstallationError* exception is raised when the *installApplication* operation detects installation failures of an application's files. (OE0268)
   1. **Pass**: The *ApplicationInstallationError* exception is raised.
   2. **Fail**: The *ApplicationInstallationError* exception is not raised.
E. Verify that the *ApplicationInstallationError* exception error number is a CF::*ErrorNumberType*. (OE0200)
   1. **Pass**: The error number is a CF::*ErrorNumberType*.
   2. **Fail**: The error number is not a CF::*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_057				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the domain manager. (OE0200, OE0261, OE0268)				
1. Locate the domain manager's source code.	**Pass**: The domain manager code is located. (OE0200, OE0261, OE0268) **Fail**: The domain manager code is not located. (OE0200, OE0261, OE0268)		The system engineer can be of great assistance in locating this code.	
2. Locate the *installApplication* operation in the domain manager source code and record the file name(s).	**Pass**: The *installApplication* operation is found in the domain manager's code. (OE0200, OE0261, OE0268) **Fail**: The *installApplication* operation is not found in the domain manager's code. (OE0200, OE0261, OE0268)			
B. Verify that the *installApplication* operation detects installation failures of an application's files.(OE0261, OE0268)				
3. Locate the code within the *installApplication* operation that actually installs the application files.	**Pass**: The install code is located. (OE0261, OE0268) **Fail**: The install code is not located. (OE0261, OE0268)			
4. Verify the source code checks for failures related to installing the application's files.	**Pass**: The code checks for installation errors. (OE0261, OE0268) **Fail**: The code checks for installation errors. (OE0261, OE0268)			
C. Verify that a FAILURE_ALARM log record is written to the domain manager's log when the *installApplication* operation detects installation failures of an application's files. (OE0261).				

OE_TC_057				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Verify in the source code that a FAILURE_ALARM log record is written to the domain manager's log.	**Pass**: A record is written to the domain manager's log. (OE0261) **Fail**: A record is not written to the domain manager's log. (OE0261)			
D. Verify that the *ApplicationInstallationError* exception is raised when the *installApplication* operation detects installation failures of an application's files. (OE0268)				
6. Verify in the source code that an *ApplicationInstallationError* exception is raised.	**Pass**: The exception is raised. (OE0268) **Fail**: The exception is not raised. (OE0268)			
E. Verify that the ApplicationInstallationError exception error number is a CF::ErrorNumberType. (OE0200)				
7. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0200) **Fail:** The exception does not include an error number (ErrorNumber) of ErrorNumberType. (OE0200)			
End of Test				

Test Recording Log – OE_TC_057					
Step 2 (File containing the *installApplication* code)	Step 4 (check for installation failure)	Step 5 (FAILURE_ALARM logged)	Step 6 (*ApplicationInstallationError* exception)	Step 7 (*ErrorNumberType)*	Notes

Test Summary OE_TC_057

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):

OE0200_____

OE0261_____

OE0268_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.12. OE_TC_065 - DomainManager :: registerService

Test Case Number: OE_TC_065

DomainManager::registerService

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0314	The *registerService* operation shall associate the input *registeringService* parameter with the input *registeredDeviceMgr* parameter in the domain manager, when the *registeredDeviceMgr* parameter indicates a device manager that is registered with the domain manager.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-46, Section 3.1.3.2.3.6.7.3

Test Objective

This test case verifies OE0314. The objective of this test case is to verify that the *registerService* operation associates its *registeringService* parameter with the device manager with which it is registered. The registered device manager that the *registeringService* must be associated with is indicated by the *registeredDeviceMgr* parameter.

Places to Verify

DomainManager

IDL References

Operations

void registerService (

in Object registeringService,

in CF::DeviceManager registeredDeviceMgr,

in string name)

raises (CF::InvalidObjectReference,

CF::DomainManager::DeviceManagerNotRegistered,

CF::DomainManager::RegisterError);

Preconditions

- The Domain Manager source code files of the Core Framework are available.

Test Description

A. Locate the *registerService* operation in the domain manager source code (OE0314).
   1. **Pass:** The *registerService* operation is found.
   2. **Fail:** The *registerService* operation is not found.

B. Verify that the *registerService* operation confirms that the registeredDeviceMgr parameter exists in the registered device managers of the domain manager. (OE0314)
   1. **Pass:** The *registerService* operation confirms that the registeredDeviceMgr parameter exists in the registered device managers of the domain manager.
   2. **Fail:** The *registerService* operation does not confirm that the registeredDeviceMgr parameter exists in the registered device managers of the domain manager.

C. Verify that the *registerService* operation associates the registeringService parameter with the device manager indicated by the registeredDeviceMgr parameter. (OE0314)
   1. **Pass:** The *registerService* operation associates the registeringService parameter with the device manager indicated by the registeredDeviceMgr parameter.
   2. **Fail:** The *registerService* operation does not associate the registeringService parameter with the device manager indicated by the registeredDeviceMgr parameter.

Semi-automated Test Steps

After running JTAP's DomainManager registerService unregisterService test, perform manual steps 1 through 3.

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_065				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. A. Locate the *registerService* operation in the domain manager source code (OE0314).				
1. Search the domain manager source code and locate all implementations of the *registerService* operation. List the file name that contains the *registerService* operation.	**Pass:** The *registerService* source code is found. Continue the test. (OE0314). **Fail:** The *registerService* source code is not found. (OE0314).		In many cases, there will be a base class virtual function and another *registerService* operation, which is the actual implementation that the domain manager uses.	
B. Verify that the *registerService* operation confirms that the registeredDeviceMgr parameter exists in the registered device managers of the domain manager. (OE0314)				
2. Verify in the source code that the *registerService* operation performs a check to confirm that the registeredDeviceMgr parameter exists in the registered device managers of the domain manager.	**Pass:** The *registerService* operation confirms that the registeredDeviceMgr parameter exists in the registered device managers of the domain manager. (OE0314). **Fail:** The *registerService* operation does *not* confirm that the registeredDeviceMgr parameter exists in the registered device managers of the domain manager. (OE0314)		The registered device managers of the domain manager are listed in the domain manager's deviceManagers attribute.	

OE_TC_065				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
C. Verify that the *registerService* operation associates the registeringService parameter with the device manager indicated by the registeredDeviceMgr parameter. (OE0314)				
3. Verify that in the source code of the domain manager that the *registerService* operation contains logic to associate the service represented by the registeringService parameter to the device manager represented by the registeredDeviceMgr parameter.	**Pass**: The *registerService* operation associates the registeringService parameter with the device manager indicated by the registeredDeviceMgr parameter. (OE0314) **Fail:** The *registerService* operation does *not* associate the registeringService parameter with the device manager indicated by the registeredDeviceMgr parameter. (OE0314)		One possible place to find a clue of how the association between the registered services and device managers is made is to check the *unregisterDeviceManager* operation.	
End of Test				

Test Recording Log – OE_TC_065		
Step 1 (List of file(s) that contain the *registerService* operation)	Step 2 (*registerService* operation confirms that the registeredDeviceMgr parameter exists in domain manager's registered device managers.) (Pass/Fail?)	Step 3 (*registerService* operation associates the registeredDeviceMgr parameter with the service represented by the registeringService parameter) (Pass/Fail?)

Test Summary OE_TC_065

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):

OE0314_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.13. OE_TC_073 - DomainManager :: registerService

Test Case Number: OE_TC_073

DomainManager::registerService

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0725	The *registerService* operation shall register the new service, indicated by the input registeringService parameter, when the previously registered service has the same name and type as the new service and the reference to the registered service refers to a nonexistent object.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-46, Section 3.1.3.2.3.6.7.3

Test Objective

This test case verifies OE0725. The objective of this test is to verify that if a new service has the same name and type as a previously registered service, but the object of the registered service does not exist, then the new service will be registered.

Places to Verify

DomainManager

IDL References

Operations

void registerService (in Object registeringService, in CF::DeviceManager registeredDeviceMgr, in string name)
raises (CF::InvalidObjectReference, CF::DomainManager::DeviceManagerNotRegistered, CF::DomainManager::RegisterError);

Preconditions

- The *DomainManager* source code files are available.

Test Description

A. Locate the source code for the *registerService* operation in the *DomainManager* (OE0725).
   1. **Untested**: The *registerService* operation source code cannot be found.

B. When the name and type of the service matches a current entry, verify that the object of the current entry is checked for existence and, if the object is nonexistent, verify that the new service is registered (OE0725).
   1. **Pass:** The new service is registered when the current service object does not exist.
   2. **Fail:** The current object is not checked for existence.
   3. **Fail**: The new object is not registered when the current object does not exist.

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_073				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the *registerService* operation in the *DomainManager* (OE0725).				
1. Locate the *registerService* operation in the source code and record the file and line number.	The *registerService* operation is located. **Untested**: The *registerService* operation source code is not found.(OE0725)			
B. When the name and type of the service matches a current entry, verify that the object of the current entry is checked for existence and, if the object is nonexistent, verify that the new service is registered (OE0725).				
2. Verify that the *registerService* operation performs a search for matching service name and type.	**Pass**: A search for matching name and type is found. (OE0725) **Fail**: No search for matching name and type is performed.(OE0725)			
3. Verify that, if a match is found, the current entry is checked for existence.	**Pass**: The current entry is checked for existence.(OE0725) **Fail**: The current entry is not checked for existence.(OE0725)			
4. Verify that if the object does not exist that the new service is registered.	**Pass**: The new service is registered. (OE0725) **Fail**: The new service is not registered. (OE0725)			
End of Test				

Test Recording Log – OE_TC_073			
Step 1 (File and line number)	Step 2 (matching name and type search)	Step 3 (existence check)	Step 4 (new service registered.)

Test Summary OE_TC_073

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):

OE0725_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.14. OE_TC_074 - ApplicationFactory :: create

Test Case Number: OE_TC_074

ApplicationFactory::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0184	The *create* operation shall create the specified event channel if the event channel does not exist.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-27, Section 3.1.3.2.2.5.1 (create) Page 3-29, Section 3.1.3.2.2.5.1.3 (actual requirement)
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-21, C-22, C-23.
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-43.

Test Objective

This test case verifies requirement OE0184. The objective of this test is to verify that if the specified event channel does not already exist, then the *create* operation will create the specified event channel.

Places to Verify

CF ApplicationFactory

IDL References

Data

typedef sequence <DeviceAssignmentType> DeviceAssignmentSequence;
typedef sequence <DataType> Properties;

Operations

CF::Application create (
in string name,
in CF::Properties initConfiguration,
in CF::DeviceAssignmentSequence deviceAssignments)

raises (CF::ApplicationFactory::CreateApplicationError,
CF::ApplicationFactory::CreateApplicationRequestError,
CF::ApplicationFactory::InvalidInitConfiguration); };

Preconditions

- The source code file(s) for the ApplicationFactory's *create* operation are available.

Test Description

A. Identify the source code file(s) for the ApplicationFactory's create operation as it relates to the existence or non-existence of the event channel. (OE0184).
   1. Locate that the ApplicationFactory's create() operation source exists.
      a. **Pass:** The ApplicationFactory's source code file(s) for *create()* is found.
      b. **Untested**: The ApplicationFactory's source code file(s) for *create()* is not found.

B. Analyze the source code implementation of create to determine that when the specified event channel does not already exist, the create operation will create the specified event channel. (OE0184).
   1. **Pass:** The *create* operation creates the specified event channel if the event channel does not already exist.
   2. **Pass:** The *create* operation does not create the specified event channel if the event channel already exists.
   3. **Fail:** The *create* operation does not create the specified event channel if the event channel does not already exist.

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_074				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code file(s) for the ApplicationFactory's create operation as it relates to the existence or non-existence of the event channel. (OE0184).				
1. Locate the source code file or files for the *create* operation within ApplicationFactory.	**Pass**: The source code file(s) for *create()* is found. (OE0184) **Untested:** The source code file(s) for *create()* is not found. (OE0184)			
B. Analyze the source code implementation of create to determine that when the specified event channel does not already exist, the create operation will create the specified event channel. (OE0184).				
2. Analyze the source code implementation to determine that when the specified event channel does not already exist, the create operation will create the specified event channel.	**Pass**: The create operation creates the specified event channel if the event channel does not already exist. (OE0184) **Fail**: The create operation does not create the specified event channel if the event channel does not already exist. (OE0184)			
End of Test				

Test Recording Log – OE_TC_074	
Step 1 (Source code file name(s))	Step 2 (If event channel not exist, create event channel?)

Test Summary OE_TC_074

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):

OE0184_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.15. OE_TC_090 - DeviceManager :: getComponentImplementationId

Test Case Number: OE_TC_090

DeviceManager::getComponentImplementationId

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0504	The getComponentImplementationId operation shall return the SPD implementation element's id attribute that matches the SPD implementation element used to create the component identified by the input componentInstantiationId parameter.
OE0505	The *getComponentImplementationId* operation shall return an empty string when the input componentInstantiationId parameter does not match the id attribute of any SPD implementation element used to create the component.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-57 / Section 3.1.3.2.4.6.6.4
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-34
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	

Test Objective

This test case verifies requirement OE0504 and OE0505. The objective of this test is to verify that the *getComponentImplementationId* operation returns the SPD implementation element's id attribute that matched the element used to create the component identified by the input parameter. Furthermore, this test verifies that the returned value of the *getComponentImplementationId* operation is an empty string when the input parameter does not match any SPD implementation element used to create the component.

Places to Verify

DeviceManager

IDL References

Operations

string getComponentImplementationId (in string componentInstantiationId);

Preconditions

- All the Domain profile files are available
- The DeviceManager source code files are available.

Test Description

These steps are to be performed for the DeviceManager(s)

A. Verify that the *getComponentImplementationId* operation is implemented. (OE0504, OE0505)
   1. **Fail:** No *getComponentImplementationId* operation exists for the DeviceManager.

B. Identify the mechanism by which the SPD implementation elements are input into the *getComponentImplementationId* operation. (OE0504, OE0505)
   1. Parsing of the IDL within the embedded system is one possible option.
   2. Pre-parsing of the IDL to create a table is another possible option.
   3. **Fail:** Hard-coded checks within the *getComponentImplementationId* operation.

C. Verify that the *getComponentImplementationId* operation compares the input *componentInstantiationId* string with values obtained from the SPD or input table. (OE0504, OE0505)
   1. **Pass:** The *getComponentImplementationId* operation performs comparisons of the input *componentInstantiationId* string.
   2. **Fail:** The *getComponentImplementationId* operation does not perform comparisons of the input *componentInstantiationId* string.

D. Review the logic in the *getComponentImplementationId* operation as a result of the comparison above.
   1. Verify that *getComponentImplementationId* operation properly handles the case for which a match is found. (OE0504)
      a. **Pass**: The returned SPD implementation element's id matches the one used to create the component identified by the input parameter.
      b. **Fail:** The returned SPD implementation element's id does not match.
   2. Verify that the *getComponentImplementationId* operation returns an empty string when no match is found. (OE0505)
      a. **Pass:** An empty string is returned when no match is found.
      b. **Fail:** No match is found but the string is not empty.

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_090				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
These steps are to be performed for the DeviceManager(s).				
A. Verify that the *getComponentImplementationId* operation is implemented. (OE0504, OE0505)				
1. Verify that the *getComponentImplementationId* operation is implemented.	**Fail**: No *getComponentImplementationId* operation exists for the Device Manager. (OE0504, OE0505)			
B. Identify the mechanism by which the SPD implementation elements are input into the *getComponentImplementationId* operation. (OE0504, OE0505)				
2. Identify the SPD implementation elements for the DeviceManager.	**Fail**: No SPD implementation elements. (OE0504, OE0505)			
3. Identify the mechanism by which the SPD implementation elements are input into the *getComponentImplementationId* operation.	Parsing of the IDL within the embedded system is one possible option. Pre-parsing of the IDL to create a table is another possible option. **Fail**: No mechanism is found for the SPD implementation elements to be input into the *getComponentImplementationId* operation. (OE0504, OE0505)			
C. Verify that the *getComponentImplementationId* operation compares the input *componentInstantiationId* string with values obtained from the SPD or input table. (OE0504, OE0505)				

OE_TC_090				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the *getComponentImplementationId* operation compares the input *componentInstantiationId* string with values obtained from the SPD or input table.	**Pass**: A comparison is performed between the input parameter and the SPD elements within the *getComponentImplementationId* operation. (OE0504, OE0505) **Fail**: No comparison is performed between the input parameter and the SPD elements within the *getComponentImplementationId* operation. (OE0504, OE0505)			
D. Review the logic in the *getComponentImplementationId* operation as a result of the comparison above.				
1. Verify that *getComponentImplementationId* operation properly handles the case for which a match is found. (OE0504)				
5. Verify that the *getComponentImplementationId* operation returns the SPD implementation element's id attribute when a match is found.	**Pass**: The *getComponentImplementationId* operation returns the SPD implementation element's id attribute when a match is found. (OE0504) **Fail**: The *getComponentImplementationId* operation does not return the SPD implementation element's id attribute when a match is found. (OE0504)			
2. Verify that the operation returns an empty string when no match is found. (OE0505)				

OE_TC_090				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Verify that the operation returns an empty string when no match is found.	**Pass**: An empty string is returned when no match is found. (OE0505) **Fail**: Something other than an empty string is returned when no match is found. (OE0505)			
End of Test				

Test Recording Log – OE_TC_090				
Operation implemented Step 1	SPD Implementation Elements Steps 2 and 3	Compares componentInstantiationId Step 4	Returns id attribute string when a match is found Step 5	Returns empty string when no match is found Step 6

Test Summary - OE_TC_090

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)**:**

OE0504 _____

OE0505 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.16. OE_TC_108 - Application :: releaseObject releases all objects

Test Case Number: OE_TC_108

Application::releaseObject

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0137	The application shall release all object references to the components making up the application.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-23 thru 3-24, Section 3.1.3.2.1.6.1.3

Test Objective

This test case verifies OE0137. The test will verify that the *releaseObject* operation of an instantiation of the *Application* class releases all object references to the components, which make up an application. In this test case, *Application* will refer to an instantiation of the *Application* class.

Places to Verify

Application

IDL References

Operations

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

Preconditions

- Source code files with the *releaseObject* operation for the *Application* under test are available.

Test Description

For the *releaseObject* operation within the *Application* under test:

A. Locate the source code for the *releaseObject* operation. (OE0137)

1. **Pass**: The *releaseObject* operation removes all references to a component.
2. **Pass**: The *releaseObject* source code is found.

B. Verify that the *releaseObject* operation removes all references to a component after it releases them. (OE0137)
1. **Fail**: The releaseObject operation does not remove all references to a component.
2. **Pass**: The releaseObject operation removes all references to a component.

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_108				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the *releaseObject* operation (OE0137)				
1. Locate the source code for the *releaseObject* operation in the *Application* class and record the file name.	**Untested:** No *releaseObject* operation is found in the *Application* class. (OE0137) End the test.		The *releaseObject* operation is part of the *Lifecycle* class, which is inherited by the *Resource*, *Device*, and *Application* classes. We are only interested in the *Application* class for this test.	
B. Verify that the *releaseObject* operation removes all references to components after it releases them. (OE0137)				
2. Verify that all object references are released during the invocation of the component's *releaseObject*.	**Fail:** All object references are not released during the invocation of the component's *releaseObject*. (OE0137) **Pass:** All object references are released during the invocation of the component's *releaseObject*. (OE0137)		*The Application should be maintaining a list of the components of an application. We are verifying that the components are removed from this list.*	
End of Test				

Test Recording Log – OE_TC_108	
Step 1 (*Application::releaseObject* file)	**Step 2** (*releaseObject* object references released) (yes/no)

Test Summary OE_TC_108

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):

OE0137_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.17. OE_TC_112 - DeviceManager's execparam properties

Test Case Number: OE_TC_112

DeviceManager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0756	The device manager shall pass the componentinstantiation element "execparam" properties that have values as parameters.
OE0754	The device manager shall pass "execparam" parameters' IDs and values as string values.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-53, Section 3.1.3.2.4.5
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-22, Section D.4.1.1.6; Page D-51, Section D.7.1.4.1.6
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-1 through C-2, Section C.1

Test Objective

This test case verifies OE0754, and OE0756. The objective of this test is to verify that the DeviceManager(s) correctly provides the *execparam* properties as a parameter to the *ExecutableDevice::execute* operation, when deploying services. The *execparam* should contain IDs and values of type string.

Places to Verify

DeviceManager, DomainManager, and Devices

IDL References

Data

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

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 OE source code that contains the *ExecutableDevice::execute* operation is available.
- The Domain Profile files that contain the DeviceManager and execparam are available.
- The Domain Profile test has passed.

Test Description

A. Determine if the DeviceManager calls the *ExecutableDevice::execute* operation. (OE0754, OE0756)
   1. **Pass**: The DeviceManager calls the *ExecutableDevice::execute* operation.
   2. **Untested**: The DeviceManager does not call any *ExecutableDevice::execute* operation.

B. Verify that the DeviceManager passes the *execparam* IDs and values as strings. (OE0754)
   1. **Pass**: The DeviceManager passes the *execparam* IDs and values as strings.
   2. **Fail**: The DeviceManager does not pass the *execparam* IDs and values as strings.

C. Identify where the componentinstantiation element is parsed from the XML, and verify that this is supplied as *execparam* properties. (OE0756)
   1. **Pass**: The componentinstantiation element is supplied to the *ExecutableDevice::execute* operation as *execparam* properties.
   2. **Fail**: The componentinstantiation element is not supplied to the *ExecutableDevice::execute* operation as *execparam* properties.

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_112				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Determine if the DeviceManager calls the *ExecutableDevice::execute* operation. (OE0754, OE0756)				
1. Search in the source code for where the DeviceManager calls the *ExecutableDevice::execute* operation.	**Pass**: The DeviceManager calls the *ExecutableDevice::execute* operation. (OE0754, OE0756) **Untested**: The DeviceManager does not call any *ExecutableDevice::execute* operation. (OE0754, OE0756)		May need Developer support to verify where DeviceManager calls the *ExecutableDevice::execute* operation to deploy services. Note: Search for "execute". Look in files whose filenames contain a variation of the name "DeviceManager". There may be multiple DeviceManager files (red, black, etc.).	
B. Verify that the DeviceManager passes the *execparam* IDs and values as strings. (OE0754)				
2. Verify that the *execparam*, which is passed to the *ExecutableDevice::execute* operation as the third parameter, is of type CF::Properties.	**Pass**: Parameter passed to *ExecutableDevice::execute* operation is of type CF::Properties. **Fail**: Parameter passed to *ExecutableDevice::execute* operation is not of type CF::Properties.			

OE_TC_112				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *execparam*, which is passed to the *ExecutableDevice::execute* operation as a parameter, contains IDs and values as strings.	**Pass**: The DeviceManager passes the *execparam* IDs and values as strings to the *ExecutableDevice::execute* operation. (OE0754) **Fail**: The DeviceManager does not pass the *execparam* IDs and values as strings to the *ExecutableDevice::execute* operation. (OE0754)			
C. Identify where the componentinstantiation element's componentproperties are parsed from the XML, and verify that this is supplied as *execparam* properties. (OE0756)				
4. Verify that the *componentinstantiation* element's *componentproperties are* supplied to the *ExecutableDevice::execute* operation as *execparam* properties.	**Pass**: The *componentinstantiation* element's *componentproperties* are supplied to the ExecutableDevice::execute operation as *execparam* properties. (OE0756) **Fail**: The *componentinstantiation* element's *componentproperties are* not supplied to the ExecutableDevice::execute operation as *execparam* properties. (OE0756)		This step is to ensure that the parameter contains data parsed from the XML.	
End of Test				

Test Recording Log – OE_TC_112			
Step1 (Location of the source code call in the DeviceManager's execute operation)	**Step2** (The *execparam* is of type CF::Properties – Y/N)	**Step3** (The *execparam* contains IDs and values as strings – Y/N)	**Step4** (The *execparam* is componentinstantiation element – Y/N)

Test Summary OE_TC_112

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):

OE0754_____

OE0756_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.18. OE_TC_113 - DomainManager :: registerDevice

Test Case Number: OE_TC_113

DomainManager::registerDevice

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0719	The registerDevice operation shall establish any pending connections from previously registered device managers when the registering device completes these connections.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-39, Section 3.1.3.2.3.6.2.3

Test Objective

This test case verifies OE0719. The objective of this test is to verify that the DomainManager's registerDevice operation establishes connections that are pending. Pending connections are broken connections as a result of unregisterDevice operations. If the new device that is being registered meets the criteria of the needs of the pending connection then the new device will complete the connection. A pending connection can exist when a new device is registered and its connection counterpart is not yet existent in the domain.

Places to Verify

DomainManager

IDL References

Operations

```
void registerDevice (
        in CF::Device registeringDevice,
        in CF::DeviceManager registeredDeviceMgr )
raises (CF::InvalidObjectReference, CF::InvalidProfile,
        CF::DomainManager::DeviceManagerNotRegistered,
        CF::DomainManager::RegisterError);
```

Preconditions

- The OE source code files having the DomainManager's registerDevice operation are available.
- The radio set deploys devices.

Test Description

A. Verify that the DomainManager's registerDevice operation is implemented. (OE0719)
   1. **Pass**: The DomainManager's registerDevice operation is implemented.
   2. **Fail**: The DomainManager's registerDevice operation not implemented.

B. Verify that the DomainManager's registerDevice operation establishes connections that are pending. (OE0719)
   1. **Pass**: The DomainManager's registerDevice operation establishes pending connections.
   2. **Fail**: The DomainManager's registerDevice operation does not establish pending connections.

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_113				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the DomainManager's registerDevice operation is implemented. (OE0719)				
1. Find the implementation of the DomainManager's registerDevice operation. Record the file name(s).	**Pass:** The registerDevice operation implementation is found.(OE0719) **Untested:** The registerDevice operation implementation is not found. (OE0719)		Search on key word "registerDevice" void CF_DomainManager_impl::registerDevice (CF::Device_ptr pRegisteringDevice, CF::DeviceManager_ptr pRegisteredDevMgr, CORBA::Environment& _env)	
B. Verify that the DomainManager's registerDevice operation establishes connections that are pending. (OE0719)				
2. Verify in the source code that pending connections are established.	**Pass:** The registerDevice operation establishes pending connections. (OE0719) **Fail:** The registerDevice operation does not establish pending connections. (OE0719)		All connections in a "pending" state for the registering device's DeviceManager are established by the registerDevice operation. Record the names of any helper operations used to implement the pending connections functionality.	
End of Test				

Test Recording Log – OE_TC_113	
Step 1 (Files having registerDevice operation)	Step 2 (Helper function names)

Test Summary OE_TC_113

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):

OE0719_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.19. OE_TC_116 - Framework Control Interfaces

Test Case Number: OE_TC_116
Framework Control Interfaces

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0120	Framework Control Interfaces shall be implemented using the CF IDL presented in Appendix C.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-20, Section 3.1.3.2
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 OE0120. The objective of this test is to verify that the Framework Control Interfaces are implemented using the CF.idl as presented in Appendix C. The CF.idl implements the CF module. The CF.idl supplies the following interfaces:

Interface Table			
Interface	**Interfaces Inherited**	**Interface**	**Interfaces Inherited**
AggregateDevice	None	FileSystem	None
Application	Resource	LifeCycle	None
ApplicationFactory	None	LoadableDevice	Device
Device	Resource	Port	None
DeviceManager	PropertySet, PortSupplier	PortSupplier	None
DomainManager	PropertySet	PropertySet	None
ExecutableDevice	LoadableDevice	Resource	LifeCycle, TestableObject, PropertySet, PortSupplier
File	None	ResourceFactory	None
FileManager	FileSystem	TestableObject	None

Places to Verify

Developer IDL and source code

IDL References

None

Preconditions

- The CF source code files are available.
- The developer's equivalent of the CF.idl file is available.
- A file comparison utility, that outputs the differences between two files, is available.

Test Description

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. (OE0120)
   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 Framework Control Interfaces was generated from the CF.idl. (OE0120)
   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 each CF interface implemented by the OE inherits from the base classes in the skeleton code generated from the CF.idl. (OE0120)
   1. **Pass**: All CF interfaces inherit from the base classes generated from the CF.idl.
   2. **Fail**: At least one CF interface does not inherit from the base classes generated from the CF.idl.

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_062.

OE_TC_116				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
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. (OE0120).				
1. Locate the CF.idl supplied by the developer and record its location.	**Pass**: The CF.idl is located. (OE0120) **Fail**: The CF.idl is not located. (OE0120)			
2. Using a diff tool, compare the CF.idl supplied with the SCA CF.idl from attachment 1 of appendix C of the SCA	**Pass**: The developer supplied CF.idl and the SCA CF.idl match. (OE0120) **Fail**: The CF.idl files do not match. (OE0120)		White space 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. Verify that the skeleton code for Framework Control Interfaces (Application, ApplicationFactory, DeviceManager, and DomainManager) was generated from the CF.idl. (OE0120)				
3. Search the supplied source code header files for "namespace CF" and verify that it is found.	**Pass**: "namespace CF" is found. (OE0120) **Fail**: "namespace CF" is not found. (OE0120)		For C++ a module in an idl translates to a namespace. For Java it would be a package. For Ada it would be a package. The search string will have to be modified appropriately.	
For each interface from the Interface table in the Test Objective section, perform the following steps:				

OE_TC_116				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Search for "class <interface name>" in the appropriate header files.	**Pass**: "class <interface name>" is found. (OE0120) **Fail**: "class <interface name>" is not found. (OE0120)		As an example, for the device interface, search for "class device". For Java, an interface maps to a signature and an operation interface. For Ada, an interface maps to a child package. The search string will have to be modified appropriately.	
5. If an interface inherits from another interface, then verify that it is a sub class of the interface it inherits.	**Pass**: The interface does not inherit and it is not a subclass. (OE0120) **Pass**: The interface does inherit and the class is a subclass. (OE0120) **Fail**: The interface inherits but the class is not a subclass. (OE0120) . **Fail**: The interface does not inherit but the class is a subclass. (OE0120)		As an example, device inherits from resource, so the code should look like: class Device : public virtual Resource Java "extends" a class. Ada implements it as a "new" of the interface it inherits. The search string will have to be modified appropriately.	
C. Verify that each CF interface implemented by the OE inherits from the base classes in the skeleton code generated from the CF.idl. (OE0120)				
6. Identify every component of the OE that implements a CF interface	Components that implement CF interfaces are identified.		The Spectra tool is of great assistance in performing this identification.	
7. Verify that the component interface implementation inherits from the base classes generated from the CF.idl.	**Pass**: The interface implementation inherits from the base classes. (OE0120) **Fail**: The interface implementation does not inherit from the base classes. (OE0120)		Steps 4 and 5 verify that the base interfaces are implemented using the idl. This step verifies that every device, resource, etc. inherits from the base interfaces.	
End of Test				

Test Recording Log – OE_TC_116				
Step1 (CF.idl location)				
Step2 (matches)				
Step3 (IDL module maps properly – header file name)	**Step4** (IDL interface maps properly)	**Step5** (IDL interface inherits properly)	**Step6** (Components identified)	**Step7** (base classes inherited)

Test Summary OE_TC_116

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):

OE0120_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.20. OE_TC_121 - ApplicationFactory :: create does property comparisons

Test Case Number: OE_TC_121

ApplicationFactory::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0707	The *create* operation shall perform the comparison of allocation properties of the application to those of each candidate device, according to the allocation property's *action* element, for those application component properties whose *kindtype* is *allocation* and whose *action* element is not *external*.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-27 through 3-31, Section 3.1.3.2.2.5.1.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-2, C-20
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Page D-19 through D-27, Section D.4.1

Test Objective

This test case verifies OE0707. The objective of this test is to verify that property comparisons are performed in accordance with the data in the Domain Profile files. The possible values for the action element, other than external, are in the following table.

Value	Meaning	Value	Meaning	Value	Meaning
eq	equal to	gt	greater than	ge	greater than or equal to
ne	not equal to	lt	less than	le	less than or equal to

Places to Verify

ApplicationFactory

IDL References

Data

```
struct DataType {
    string id;
    any value;
```

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

struct DeviceAssignmentType  {
  string componentId;
  string assignedDeviceId;
};
typedef sequence <DeviceAssignmentType> DeviceAssignmentSequence;
```

Operations

```
CF::Application create ( in string name, in CF::Properties initConfiguration, in CF::DeviceAssignmentSequence deviceAssignments )
    raises (CF::ApplicationFactory::CreateApplicationError,
            CF::ApplicationFactory::CreateApplicationRequestError,
            CF::ApplicationFactory::InvalidInitConfiguration)
```

Preconditions

- The Domain profile files are available.
- The *ApplicationFactory* source code files are available.

Test Description

A. Locate the source code that parses the Domain Profile files. (OE0707)
   1. **Untested**: Domain Profile files are not parsed.
B. Determine how the allocation property kindtype and action elements are saved. (OE0707)
   1. **Untested**: The kindtype and action elements are not saved.
C. Locate the source code for the *ApplicationFactory create* operation. (OE0707)
   1. **Untested**: No source code for the *ApplicationFactory create* operation is available.
D. Verify that the *create* operation performs a comparison of properties, based on the action element of the property. (OE0707)
   1. **Pass:** Comparisons are performed properly.
   2. **Fail:** No comparisons are performed.
   3. **Fail:** Comparisons are not based on the action element.

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_121				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code that parses the Domain Profile files. (OE0707)				
1. Locate the source code that parses the Domain Profile files and record the file name.	The source code that parses the Domain Profile files is found. (OE0707) **Untested**: The source code that parses the Domain Profile files is not found. (OE0707)		This code may not be delivered. On some SDR's this is done externally to the SDR. In this case, determine how the data parsed externally is retrieved	
B. Determine how the allocation property kindtype and action elements are saved. (OE0707)				
2. Determine how and record where the kindtype and action elements of the allocation properties are saved.	How and where the kindtype and action elements of the allocation properties are saved is determined. **Untested**: How and where the kindtype and action elements of the allocation properties are saved cannot be determined. (OE0707)		The developer's engineer may have to supply this information.	
C. Locate the source code for the *ApplicationFactory create* operation. (OE0707)				
3. Locate the source code for the *create* operation of the *ApplicationFactory* and record the file name.	**Untested**: No source code for the *create* operation can be found. (OE0707)			
D. Verify that the *create* operation performs a comparison of properties, using the data as saved in step 2. (OE0707)				

OE_TC_121				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the *create* operation does a comparison of the *allocation* properties.	**Pass**: A comparison of the *allocation* properties is performed. (OE0707) **Fail**: A comparison of the *allocation* properties is not performed (OE0707)		Step 2 recorded where this data was saved.	
5. Verify that the *create* operation uses the *action* type to determine how to do the comparison.	**Pass**: The action type is used to determine the type of comparison done. (OE0707) **Fail**: The action type is not used to determine the type of comparison done. (OE0707)		The possible *action* types are in the table in the Test Objective. Step 2 recorded where this information was saved.	
End of Test				

Test Recording Log – OE_TC_121			
Step2 (property name and action type)	**Step3** (*create* operation source code file)	**Step4** (Comparison performed)	**Step5** (*action* type used)

Test Summary OE_TC_121

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):

OE0707_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.21. OE_TC_122 - DomainManager :: uninstallApplication raises ApplicationUninstallationError

Test Case Number: OE_TC_122

DomainManager::uninstallApplication raises ApplicationUninstallationError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0309	The *uninstallApplication* operation shall raise the ApplicationUninstallationError exception when an internal error causes an unsuccessful uninstallation of the application.
OE0203	The error number shall indicate a CF ErrorNumberType value.
OE0301	The uninstallApplication operation shall, upon unsuccessful uninstall of an application, write a FAILURE_ALARM log record to a domain manager's log.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-45, Section 3.1.3.2.3.6.6.3; Page 3-45, Section 3.1.3.2.3.6.6.5; Page 3-34, Section 3.1.3.2.3.3.9; Page 3-94, Section 3.1.3.6.1.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-3, C-17, C-20, Section C.1

Test Objective

This test case verifies requirement OE0301, OE0309 and OE0203. The objective of this test is to verify that the *uninstallApplication* operation provided by the domain manager writes a FAILURE_ALARM log record to a domain manager's log and raises the ApplicationUninstallationError exception when an internal error prohibits the successful uninstallation of the application. The exception should also be accompanied with a CF::ErrorNumberType value (OE0203).

Places to Verify

DomainManager

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_EMSGSIZE, 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 ApplicationUninstallationError {CF::ErrorNumberType errorNumber; string msg;};

Operations

void uninstallApplication(in string applicationId)
raises (CF::DomainManager::InvalidIdentifier, CF::DomainManager::ApplicationUninstallationError);

Preconditions

- The domain manager source code files of the Core Framework are available.

Test Description

A. Identify the implementations of the *uninstallApplication* operation in the domain manager. (OE0301, OE0309)
   1. **Fail**: The *uninstallApplication* operation does not exist in the domain manager source code.

For every implementation of the *uninstallApplication* operation in the domain manager, perform the following steps:

B. Verify that the uninstallApplication operation writes a FAILURE_ALARM log record to a domain manager's log when the operation cannot successfully uninstall the application. (OE0301)
   1. **Pass:** The *uninstallApplication* operation writes a FAILURE_ALARM log record to a domain manager log when the operation cannot successfully uninstall the application.
   2. **Fail:** The *uninstallApplication* operation does not write a FAILURE_ALARM log record to a domain manager log when the operation cannot successfully uninstall the application.

C. Verify that the *uninstallApplication* operation raises the ApplicationUninstallationError exception when the operation encounters an internal error that causes an unsuccessful uninstallation of the application. (OE0309)

1. **Pass**: The *uninstallApplication* operation raises the ApplicationUninstallationError when the operation encounters an internal error that causes an unsuccessful installation of the application.
2. **Fail**: The *uninstallApplication* operation does not raise the ApplicationUninstallationError when the operation encounters an internal error that causes an unsuccessful uninstallation of the application.

D. Verify that the error number that accompanies the ApplicationUninstallationError exception is a CF::ErrorNumberType value. (OE0203)
1. **Pass:** The ApplicationUninstallationError provides an error number with a CF::ErrorNumberType value for the error condition.
2. **Fail:** The ApplicationUninstallationError 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_122				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the *uninstallApplication* operation in the domain manager. (OE0301, OE0309)				
1. Examine the source code of the domain manager within the Core Framework and identify the implementations of the *uninstallApplication* operation. List the source code files of the implementation of the *uninstallApplication* operation.	**Fail:** The *uninstallApplication* operation does not exist in the domain manager source code. (OE0301, OE0309)			
For every implementation of the *uninstallApplication* operation in the domain manager, perform the following steps:				
B. Verify that the *uninstallApplication* operation writes a FAILURE_ALARM log record to a domain manager's log when the operation cannot successfully uninstall the application. (OE0301)				
2. Examine the source code of the *uninstallApplication* operation and verify that a FAILURE_ALARM log record is written to a domain manager log when the operation cannot successfully uninstall the application.	**Pass**: The uninstallApplication operation writes a FAILURE_ALARM log record to a domain manager log when the operation cannot successfully uninstall the application. (OE0301) **Fail**: The uninstallApplication operation does not write a FAILURE_ALARM log record to a domain manager log when the operation cannot successfully uninstall the application. (OE0301)			
C. **Verify that the *uninstallApplication* operation raises the ApplicationUninstallationError exception when the operation encounters an internal error that causes an unsuccessful uninstallation of the application. (OE0309)**				

OE_TC_122				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Examine the source code of the *uninstallApplication* operation and verify that the ApplicationUninstallationError exception is raised when the operation encounters an internal error that causes an unsuccessful uninstallation of the application.	**Pass**: The *uninstallApplication* operation raises the ApplicationUninstallationError when the operation encounters an internal error that causes an unsuccessful uninstallation of the application. (OE0309) **Fail**: The *uninstallApplication* operation does not raise the ApplicationUninstallationError when the operation encounters an internal error that causes an unsuccessful uninstallation of the application. (OE0309)			
D. Verify that the error number that accompanies the ApplicationUninstallationError exception is a CF::ErrorNumberType value. (OE0203)				
4. Determine if the exception includes an error number (ErrorNumber) of the type CF::ErrorNumberType.	**Pass:** The exception includes an error number (ErrorNumber) of ErrorNumberType. (OE0203) **Fail:** The exception does not include an error number (ErrorNumber) of ErrorNumberType. (OE0203)			
End of Test				

Test Recording Log – OE_TC_122			
Step1 (List the source code file(s) of the *uninstallApplication* operation.)	**Step2** (*uninstallApplication* op. writes FAILURE_ALARM log when operation fails.) (Pass/Fail?)	**Step3** (*uninstallApplication* operation raises the ApplicationUninstallationError exception when internal error occurs.) (Pass/Fail?)	**Step4** (The exception includes an error number of the type: CF::ErrorNumberType.) (Pass/Fail?)

Test Summary OE_TC_122

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):

OE0301 _____

OE0309 _____

OE0203 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.2.22. OE_TC_124 - DomainManager :: registerDevice raises RegisterError

Test Case Number: OE_TC_124

DomainManager::registerDevice raises RegisterError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0258	The *registerDevice* operation shall raise the *RegisterError* exception when an internal error exists which causes an unsuccessful registration.
OE0201	The error number shall indicate a CF ErrorNumberType value.
OE0248	The *registerDevice* operation shall write a FAILURE_ALARM log record to a domain manager log when the RegisterError exception is raised.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-39, Section 3.1.3.2.3.6.2.3 Page 3-40, Section 3.1.3.2.3.6.2.5 Page 3-33, Section 3.1.3.2.3.3.7
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-17 thru C-19

Test Objective

This test case verifies OE0248, OE0258 and OE0201. The objective of this test is to verify that the *RegisterError* is raised when an internal error causes an unsuccessful registration during the *DomainManager::registerDevice* operation, and verify that the error number value in the exception is of the type *ErrorNumberType*. This test case also verifies OE0248, which means a FAILURE_ALARM record is written to the domain manager log when the *RegisterError* exception is raised.

Places to Verify

Domain Manager

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_EMSGSIZE, 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 RegisterError { ErrorNumberType errorNumber; string msg; };
exception InvalidObjectReference { string msg; };
exception InvalidProfile { };
exception DeviceManagerNotRegistered { };

Operations

void registerDevice (in Device registeringDevice, in DeviceManager registeredDeviceMgr)
raises (InvalidObjectReference, InvalidProfile, DeviceManagerNotRegistered, RegisterError);

Preconditions

- The source code of the Core Framework is available.

Test Description

A. Identify the source code files that implement the *DomainManager::registerDevice* operation. (OE0258)
   1. **Untested**: The source code files having the implementation of *registerDevice* are not available.

For each *DomainManager::registerDevice* operation found within the OE under test, perform the following steps:

B. Verify that the *registerDevice* operation raises the *RegisterError* exception when an internal error causes an unsuccessful registration. (OE0258)
   1. **Pass**: The *registerDevice* operation raises the *RegisterError* exception.
   2. **Fail:** The *registerDevice* operation does not raise the *RegisterError* exception.

C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0201)
   1. **Pass:** The *RegisterError* exception provides an error number with an ErrorNumberType value for the error condition.
   2. **Fail:** The *RegisterError* exception does not provide an error number with an ErrorNumberType value for the error condition.

D. Verify that a FAILURE_ALARM log record is written to the domain manager's log when the *RegisterError* exception is raised. (OE0248).

1. **Pass:** A FAILURE_ALARM record is written to the domain manager's log.
2. **Fail:** No FAILURE_ALARM record is written to the domain manager's log.

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_124				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *DomainManager::registerDevice* operation. (OE0258)				
1. Perform a keyword search for *registerDevice* operation on all source code provided by the developer. Record the file name.	**Untested:** The source code files having the implementation of *registerDevice* are not available. (OE0258)		May need the help of the software engineer to locate the source code files directories.	
B. Verify that the *registerDevice* operation raises the *RegisterError* exception when an internal error causes an unsuccessful registration. (OE0258)				
2. Examine the *registerDevice* operation where a Device is registered and determine that the *RegisterError* exception is raised when an internal error exists causes an unsuccessful registration.	**Pass**: The *registerDevice* operation raises the *RegisterError* exception. (OE0258) **Fail:** The *registerDevice* operation does not raise the *RegisterError* exception. (OE0258)			
C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0201)				
3. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0201) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0201)			
D. Verify that a FAILURE_ALARM log record is written to the domain manager's log when the *RegisterError* exception is raised. (OE0248).				

OE_TC_124				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify in the source code that a FAILURE_ALARM log record is written to the domain manager's log.	**Pass**: A FAILURE_ALARM record is written to the domain manager's log. (OE0248) **Fail**: A FAILURE_ALARM record is not written to the domain manager's log. (OE0248)			
End of Test				

Test Recording Log – OE_TC_124			
Step1 (source file name)	**Step2** (exception raised – Y/N)	**Step3** (error number)	**Step4** (FAILURE_ALARM record)

Test Summary OE_TC_124

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):

OE0258_____

OE0201 _____

OE0248 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.23. OE_TC_125 - ApplicationFactory :: create raises CreateApplicationError

Test Case Number: OE_TC_125

ApplicationFactory::create raises CreateApplicationError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0197	The create operation shall raise the *CreateApplicationError* exception when the create request is valid but the application cannot be successfully instantiated due to internal processing error(s).
OE0152	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-26, Section 3.1.3.2.2.3.2, Page 3-31, Section 3.1.3.2.2.5.1.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 to C-4, Sections C-22 to C-23

Test Objective

This test case verifies OE0197 and OE0152. The objective of this test is to verify that the ApplicationFactory *create* operation raises the *CreateApplicationError* exception when a *create* requests is valid but the application cannot be instantiated due to internal processing error(s). Furthermore, the test case verifies that the exception contains a message and an error number of the type CF::ErrorNumberType.

Places to Verify

ApplicationFactory

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_EMSGSIZE, 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 CreateApplicationError {
        CF::ErrorNumberType errorNumber;
        string msg; };
```

Operations

```
CF::Application create (
        in string name,
        in CF::Properties initConfiguration,
        in CF::DeviceAssignmentSequence deviceAssignments )
raises (CF::ApplicationFactory::CreateApplicationError,
    CF::ApplicationFactory::CreateApplicationRequestError,
    CF::ApplicationFactory::InvalidInitConfiguration);  };
```

Preconditions

- The ApplicationFactory source code files are available.

Test Description

A. Identify the source code files that implement the ApplicationFactory *create* operation. (OE0152, OE0197)
   1. **Untested**: The ApplicationFactory *create* operation source code files are not available.

For each ApplicationFactory *create* operation found within the OE under test do the following steps:

B. Verify that the ApplicationFactory *create* operation raises the *CreateApplicationError* exception when the *create* request is valid but the application cannot be instantiated due to internal processing errors (OE0197).
   1. **Pass:** The *create* operation raises the *CreateApplicationError* exception.
   2. **Fail:** The *create* operation does not raise the *CreateApplicationError* exception.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0152)
   1. **Pass:** The *CreateApplicationError* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The *CreateApplicationError* 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_125				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the ApplicationFactory *create* operation. (OE0152, OE0197)				
1. Perform a keyword search for *create* operation on all ApplicationFactory source code provided by the developer.	The directories are recorded. **Untested:** No results from keyword search. There is no implementation of the ApplicationFactory *create* operation found. (OE0152, OE0197)		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 *create* operation implementation. Record file names of the source code where *create* operation implementations were found.	The implementations of the *create* operation are identified in the source code, and the file names of the source code are recorded.			
For each ApplicationFactory *create* operation found within the OE under test do the following steps				
B. Verify that the ApplicationFactory *create* operation raises the *CreateApplicationError* exception when the *create* request is valid but the application cannot be instantiated due to internal processing errors (OE0197).				
3. Search within the *create* operation for where the *CreateApplicationError* exception is mentioned.	Identify in the source code where create operation raises the *CreateApplicationError* exception.			
For each instance where the *CF::CreateApplicationError* exception is raised do step 4				

OE_TC_125				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Determine that *CreateApplicationError* exception is raised when the *create* request is valid but the application cannot be instantiated due to internal processing error(s).	**Pass:** The *create* operation raises the CF:: *CreateApplicationError* when the create request is valid but the application cannot be instantiated due to internal processing errors. (OE0197) **Fail:** The *create* operation does not raise the CF:: *CreateApplicationError* when the create request is valid but the application cannot be instantiated due to internal processing errors. (OE0197)		There are two other exceptions which are allowed to be raised during the *create* operation. Both are related to the validation of input parameters. The *CreateApplicationRequestError* exception must be raised "when the input CF *DeviceAssignmentSequence* parameter contains one (1) or more invalid application component to device assignment(s)." The *InvalidInitConfiguration* exception must be raised "when the input *initConfiguration* parameter is invalid." All other reasons for raising an exception during the *create* operation shall be considered internal processing errors and require the exception of this test case to be raised.	
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0152)				
5. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0152) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0152)			
End of Test				

Test Recording Log – OE_TC_125				
Step1 (Directory name location)	**Step2** (Source file name)	**Step4** (Exception raised) Y/N	**Step5** (Correct error number type) Y/N	**Notes**

Test Summary OE_TC_125

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):

OE0197_____

OE0152 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.24. OE_TC_126 - DomainManager :: registerDeviceManager raises RegisterError

Test Case Number: OE_TC_126

DomainManager::registerDeviceManager raises RegisterError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0241	The *registerDeviceManager* operation shall raise the *RegisterError* exception when an internal error exists which causes an unsuccessful registration.
OE0201	The error number shall indicate a CF ErrorNumberType value.
OE0233	The registerDeviceManager operation shall, upon unsuccessful device manager registration, write a FAILURE_ALARM log record to a domain manager's Log.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-36 through 3-39, Section 3.1.3.2.3.6.1 Page 3-33, Section 3.1.3.2.3.3.3.7
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	C-17 thru C-19

Test Objective

This test case verifies OE0233, OE0241 and OE0201. The objective of this test is to verify that the *RegisterError* is raised when an internal error exists causing an unsuccessful registration during the *registerDeviceManager* operation, and to verify that the error number value in the exception is of the type ErrorNumberType. This test also verifies that a FAILURE_ALARM log record is written to a domain manager's log when the *RegisterError* exception is raised.

Places to Verify

Domain Manager

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_EMSGSIZE, 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 InvalidObjectReference { string msg; };
exception InvalidProfile { };
exception RegisterError { ErrorNumberType errorNumber; string msg; };

Operations

void registerDeviceManager (in CF::DeviceManager deviceMgr)
raises (CF::InvalidObjectReference, CF::InvalidProfile, CF::DomainManager::RegisterError);

Preconditions

- The source code of the Core Framework is available.

Test Description

A. Identify the source code files that implement the *registerDeviceManager* operation. (OE0201, OE0233, OE0241)
   1. **Untested**: The source code files having the implementation of *registerDeviceManager* are not available.

For each *registerDeviceManager* operation found within the OE under test, perform the following steps:

B. Verify that the *registerDeviceManager* operation raises the *RegisterError* exception when an internal error causes an unsuccessful registration. (OE0241)
   1. **Pass**: The *registerDeviceManager* operation raises the *RegisterError* exception.
   2. **Fail:** The *registerDeviceManager* operation does not raise the *RegisterError* exception.

C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0201)
   1. **Pass:** The *RegisterError* exception provides an error number with an ErrorNumberType value for the error condition.
   2. **Fail:** The *RegisterError* exception does not provide an error number with an ErrorNumberType value for the error condition.

D. Verify that a FAILURE_ALARM log message is sent to a domain manager's log when the *RegisterError* exception is raised. (OE0233)
   1. **Pass:** A FAILURE_ALARM log message is sent to a domain manager's log.
   2. **Fail:** A FAILURE_ALARM log message is not sent to a domain manager's log.

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_126				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *registerDeviceManager* operation. (OE0201, OE0233, OE0241)				
1. Perform a keyword search for *registerDeviceManager* operation on all source code provided by the developer. Examine these source code files for the *registerDeviceManager* operation implementation. Record the file name(s).	**Untested**: The source code files having the implementation of *registerDeviceManager* are not available. (OE0201, OE0233, OE0241)		May need the help of the software engineer to locate the source code files directories.	
For each *registerDeviceManager* operation found within the OE under test, perform the following steps				
B. Verify that the *registerDeviceManager* operation raises the *RegisterError* exception when an internal error causes an unsuccessful registration. (OE0241)				
2. Search within the *registerDeviceManager* operation where the device manager, its device(s), and its services are registered.	The source code where *registerDeviceManager* registers a device manager, its device(s), and its services is identified.			
3. Verify that the *RegisterError* is raised when an internal error causes an unsuccessful registration.	**Pass**: The *registerDeviceManager* operation raises the *RegisterError* exception. (OE0241) **Fail:** The *registerDeviceManager* operation does not raise the *RegisterError* exception. (OE0241)			
C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0201)				

OE_TC_126				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Determine if the exception includes an error number of ErrorNumberType.	**Pass:** The exception includes an error number of ErrorNumberType. (OE0201) **Fail:** The exception does not include an error number of ErrorNumberType. (OE0201)			
D. Verify that a FAILURE_ALARM log message is sent to a domain manager's log when the RegisterError exception is raised. (OE0233)				
5. Verify that if the *RegisterError* error exception is raised, a FAILURE_ALARM log record is written to a domain manager's log.	**Pass:** A FAILURE_ALARM log record is written to a domain manager's log. (OE0233) **Fail:** A FAILURE_ALARM log record is not written to a domain manager's log. (OE0233)		FAILURE_ALARM is log level 2	
End of Test				

Test Recording Log – OE_TC_126			
Step 1 (source file name)	**Step 3** (exception raised – Y/N?)	**Step 4** (error number translated to ErrorNumberType – Y/N?)	**Step 5** (FAILURE_ALARM log written – Y/N?)

Test Summary OE_TC_126

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):

OE0241_____

OE0201 _____

OE0233 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.25. OE_TC_127 - DomainManager :: registerService raises RegisterError

Test Case Number: OE_TC_127

DomainManager::registerService raises RegisterError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0317	The *registerService* operation shall, upon unsuccessful service registration, write a FAILURE_ALARM log record to a domain manager's log.
OE0326	The *registerService* operation shall raise the *RegisterError* exception when an internal error exists which causes an unsuccessful registration.
OE0201	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-46, 3-47, Section 3.1.3.2.3.6.7 Page 3-33, Section 3.1.3.2.3.3.7
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-17 thru C-20

Test Objective

This test case verifies OE0317, OE0326 and OE0201. The objective of this test is to verify that when an internal error exists which causes an unsuccessful registration during the *registerService* operation, a FAILURE_ALARM log record is written to the domain manager's log, the *RegisterError* exception is raised and the error number value in the exception is of the type *ErrorNumberType*.

Places to Verify

Domain Manager

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_EMSGSIZE, 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 RegisterError { ErrorNumberType errorNumber; string msg; };

Operations

void registerService (in Object registeringService, in DeviceManager registeredDeviceMgr, in string name)
raises (InvalidObjectReference, DeviceManagerNotRegistered, RegisterError);

Preconditions

- The source code of the Core Framework is available.

Test Description

A. Identify the source code files that implement the *registerService* operation.
   1. **Untested**: The source code files having the implementation of *registerService* are not available. (OE0326)

For each *registerService* operation found within the OE under test, perform the following steps:

B. Verify that the *registerService* operation raises the *RegisterError* exception when an internal error occurs. (OE0326)
   1. **Pass**: The *registerService* operation raises *RegisterError* exception.
   2. **Fail:** The *registerService* operation does not raise *RegisterError* exception.

C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0201)
   1. **Pass:** The *RegisterError* exception provides an error number with an ErrorNumberType value for the error condition.
   2. **Fail:** The *RegisterError* exception does not provide an error number with an ErrorNumberType value for the error condition.

D. Verify that a FAILURE_ALARM log message is sent to the domain manager's log recording the unsuccessful service registration. (OE0317)
   1. **Pass:** A FAILURE_ALARM log message is sent to the domain manager's log.
   2. **Fail:** A FAILURE_ALARM log message is not sent to the domain manager's log.

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_127				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *registerService* operation.				
1. Perform a keyword search for *registerService* operation on the Core Framework source code provided by the developer.	**Untested**: The source code files having the implementation of *registerService* are not available. (OE0326)		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 *registerService* operation implementation. Record the file name.	The *registerService* operation for each implementation is identified and recorded.			
For each *registerService* operation found within the OE under test, perform the following steps:				
B. Verify that the *registerService* operation raises the *RegisterError* exception when an internal error occurs. (OE0326)				
3. Search within the *registerService* operation is used to register a service for a specific device manager with the domain manager.	The source code where *registerService* registers a service for a specific device manager with the domain manager is identified.			
4. Determine that the *RegisterError* is raised when an internal error exists which causes an unsuccessful registration.	**Pass**: The *registerService* operation raises the *RegisterError*. (OE0326) **Fail:** The *registerService* operation does not raise the *RegisterError*. (OE0326)			
C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0201)				

OE_TC_127				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0201) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0201)			
D. Verify that a FAILURE_ALARM log message is sent to the domain manager's log recording the unsuccessful service registration. (OE0317)				
6. Determine that a FAILURE_ALARM log message is sent to the domain manager's log recording the unsuccessful service registration.	**Pass:** A FAILURE_ALARM log message is sent to the domain manager's log. (OE0317) **Fail:** A FAILURE_ALARM log message is not sent to the domain manager's log. (OE0317)			
End of Test				

Test Recording Log – OE_TC_127			
Step2 (source file name)	**Step4** (exception raised-Y/N?)	**Step5** (ErrorNumberType-Y/N?)	**Step6** (FAILURE_ALARM log message sent-Y/N?)

Test Summary OE_TC_127

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):

OE0317_____

OE0326_____

OE0201_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.26. OE_TC_128 - DomainManager :: unregisterDeviceManager raises UnregisterError

Test Case Number: OE_TC_128

DomainManager::unregisterDeviceManager raises UnregisterError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0278	The *unregisterDeviceManager* operation shall, upon unsuccessful unregistration of a device manager, write a FAILURE_ALARM log record to a domain manager's log.
OE0285	The *unregisterDeviceManager* operation shall raise the UnregisterError exception when an internal error exists which causes an unsuccessful unregistration.
OE0202	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-43, Section 3.1.3.2.3.6.4; Page 3-44, Section 3.1.3.2.3.6.4.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3, C-19 through C-20

Test Objective

This test case verifies OE0278, OE0285 and OE0202. The objective of this test is to verify that when an internal error causes an unsuccessful unregistration (1) the *unregisterDeviceManager* operation writes a FAILURE_ALARM log record to a domain manager's log, (2) the *unregisterDeviceManager* operation raises the UnregisterError exception, (3) there is an error number as part of the raised exception, which is of the type CF::ErrorNumberType.

Places to Verify

DomainManager

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_EMSGSIZE, 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 UnregisterError {

ErrorNumberType errorNumber;

string msg;

};

Operations

void unregisterDeviceManager(
in CF::DeviceManager deviceMgr
)
raises (CF::InvalidObjectReference, CF::DomainManager::UnregisterError);

Preconditions

- The source code of the Core Framework is available.

Test Description

A. Identify the source code files that implement the *unregisterDeviceManager* operation. (OE0285, OE0278)
   1. **Pass**: The source code file(s) of the *unregisterDeviceManager* operation are identified.
   2. **Fail**: The source code file(s) of the *unregisterDeviceManager* operation are not identified.

For each *unregisterDeviceManager* operation found within the OE under test, perform the following steps:

B. Verify that a FAILURE_ALARM log message is written to the domain manager's log record upon unsuccessful unregistration of a device manager. (OE0278)
   1. **Pass:** A FAILURE_ALARM log message is written to the domain manager's log.
   2. **Fail:** A FAILURE_ALARM log message is not written to the domain manager's log.

C. Verify that the *unregisterDeviceManager* operation raises the *CF::*UnregisterError when an internal error exists which causes an unsuccessful unregistration. (OE0285)

1. **Pass**: The *unregisterDeviceManager* operation raises *CF::UnregisterError*.
2. **Fail:** The *unregisterDeviceManager* operation does not raise *CF::UnregisterError*.

D. Verify that the exception includes an error number (errorNumber) of the type ErrorNumberType. (OE0202)
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_128				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *unregisterDeviceManager* operation. (OE0285, OE0278)				
1. Perform a keyword search for *unregisterDeviceManager* operation on all source code provided by the developer. Record the file name.	**Pass**: The source code file(s) of the *unregisterDeviceManager* operation are identified. (OE0285, OE0278) **Fail**: The source code file(s) of the *unregisterDeviceManager* operation are not identified. (OE0285, OE0278)		May need the help of the software engineer to locate the source code files.	
B. **Verify that a FAILURE_ALARM log message is sent to the domain manager's log record upon unsuccessful unregistration of a device manager. (OE0278)**				
2. Determine that a FAILURE_ALARM log message is written to the domain manager's log record that records the unsuccessful unregistration of a device manager.	**Pass:** A FAILURE_ALARM log message is written to the domain manager's log with the unsuccessful unregistration of a device manager. (OE0*278*) **Fail:** A FAILURE_ALARM log message is not written to the domain manager's log with the unsuccessful unregistration of a device manager. (OE0*278)*			
C. Verify that the *unregisterDeviceManager* operation raises the *CF::*UnregisterError when an internal error exists which causes an unsuccessful unregistration. (OE0285)				
3. Search within the *unregisterDeviceManager* operation for where the *CF::UnregisterError* exception is mentioned.	Identified the *unregisterDeviceManager* operation source code where the *unregisterDeviceManager* operation raises the *CF::UnregisterError* exception.			

OE_TC_128				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Determine that *CF::UnregisterError* is raised when an internal error exists which causes an unsuccessful unregistration.	**Pass**: The *unregisterDeviceManager* operation raises the *CF::UnregisterError*. (OE0285) **Fail:** The *unregisterDeviceManager* operation does not raise the *CF::UnregisterError*. *(*OE0285)			
D. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0202)				
5. Determine if the exception includes an error number (*errorNumber*) of the type *ErrorNumberType*.	**Pass**: The exception includes an error number (errorNumber) of ErrorNumberType. (OE0202) **Fail**: The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0202)			
End of Test				

Test Recording Log – OE_TC_128				
Step 1 (source file name)	**Step 2** (log message: Y/N)	**Step 4** (exception raised: Y/N)	**Step 5** (error number: Y/N)	**Notes**

Test Summary OE_TC_128

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):

OE0278_____

OE0285_____

OE0202_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.27. OE_TC_129 - DomainManager :: unregisterDevice raises UnregisterError

Test Case Number: OE_TC_129

DomainManager::unregisterDevice raises UnregisterError

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0290	The *unregisterDevice* operation shall, upon unsuccessful unregistration of a device, write a FAILURE_ALARM log record to a domain manager's log.
OE0297	The *unregisterDevice* operation shall raise the *UnregisterError* exception when an internal error exists which causes an unsuccessful unregistration.
OE0202	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-44- to 3-45, Section 3.1.3.2.3.6.5.3 and 5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 and C-4 ; C-20

Test Objective

This test case verifies OE0297, OE0202 and OE0290. The objective of this test is to verify that, when an internal error causes an unsuccessful unregistration, (1) the *unregisterDevice* operation raises the *UnregisterError* exception (2) there is an appropriate error number that is of the type CF::ErrorNumberType, and (3) a FAILURE_ALARM log record is sent to the domain manager's log.

Places to Verify

DomainManager, unregister Device operation

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_EMSGSIZE, 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 UnregisterError {

ErrorNumberType errorNumber;

string msg; };

Operations

void unregisterDevice (

in CF::Device unregisteringDevice)

raises (CF::InvalidObjectReference, CF::DomainManager::UnregisterError);

Preconditions

- The DomainManager source code of the Core Framework is available.

Test Description

A. Identify the Domain Manager's source code files that implement the *unregisterDevice* operation. (OE0290, OE0297)
   1. **Untested**: The Domain Manager's source code files of the *unregisterDevice* operation are not available.

For each *unregisterDevice* operation found within the OE under test, perform the following steps:

B. Verify that the *unregisterDevice* operation raises the *CF::UnregisterError* when an internal error exists which causes an unsuccessful unregistration. (OE0297)
   3. **Pass**: The *unregisterDevice* operation raises *CF::UnregisterError*.
   4. **Fail:** The *unregisterDevice* operation does not raise *CF::UnregisterError*.

C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0202)
   1. **Pass**: The exception includes an error number (errorNumber) of ErrorNumberType.
   2. **Fail**: The exception does not include an error number (errorNumber) of ErrorNumberType.

D. Verify that a FAILURE_ALARM log record is sent to the domain manager's log upon unsuccessful unregistration of a device. (OE0290)
   1. **Pass**: A FAILURE_ALARM log record is sent to the domain manager's log.
   2. **Fail:** A FAILURE_ALARM log record is not sent to the domain manager's log.

Semi-automated Test Steps

After running JTAP's DomainManager unregisterDevice InvalidObjectReference test, perform manual step 6.

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_129				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *unregisterDevice* operation. (OE0297)				
1. Perform a keyword search for *unregisterDevice* operation on all source code provided by the developer.	**Untested:** The source code files of the *unregisterDevice* operation are not available. (OE0297)		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 *unregisterDevice* operation implementation. Record the file name.	The *unregisterDevice* operation for each implementation is identified and recorded.			
For each unregisterDevice operation found within the OE under test, perform the following steps:				
B. Verify that the *unregisterDevice* operation raises the *CF::UnregisterError* when an internal error exists which causes an unsuccessful unregistration. (OE0297)				
3. Search within the *unregisterDevice* operation for where the *CF::UnregisterError* exception is mentioned.	Identified the source code where the *CF::UnregisterError* exception is mentioned.			
4. Determine that *CF::UnregisterError* is raised when an internal error exists which causes an unsuccessful unregistration.	**Pass**: The *unregisterDevice* operation raises the *CF::UnregisterError*. (OE0297) **Fail:** The *unregisterDevice* operation does not raise the *CF::UnregisterError*. (OE0297)			

OE_TC_129				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0202)				
5. Determine if the exception includes an error number (*errorNumber*) of the type *ErrorNumberType*.	**Pass**: The exception includes an error number (errorNumber) of ErrorNumberType. (OE0202) **Fail**: The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0202)			
D. Verify that a FAILURE_ALARM log record is sent to the domain manager's log upon unsuccessful unregistration of a device. (OE0 290)				
6. Determine that FAILURE_ALARM log record sent to the domain manager's log that records the unsuccessful unregistration of a device.	**Pass:** A FAILURE_ALARM log record sent to the domain manager's log with the unsuccessful unregistration of a device. (OE0290) **Fail:** A FAILURE_ALARM log record not sent to the domain manager's log with the unsuccessful unregistration of a device. (OE0290)			
End of Test				

Test Recording Log – OE_TC_129				
Step2 (source file name)	**Step4** (exception raised)	**Step5** (error number)	**Step6** (FAILURE_ALARM log record)	**Notes**

Test Summary OE_TC_129

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):

OE0290_____

OE0297_____

OE0202_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.28. OE_TC_132 - DomainManager :: installApplication raises InvalidFileName

Test Case Number: OE_TC_132

DomainManager::installApplication raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0269	The *installApplication* operation shall raise the CF *InvalidFileName* exception when the input SAD file or any of the SAD's referenced filenames do not exist in the file system identified by the absolute path of the input *profileFileName* parameter.
OE0270	The *installApplication* operation shall log a FAILURE_ALARM log record to a domain manager's Log with a message consisting of "installApplication::invalid file is xxx", where "xxx" is the input or referenced filename, when the CF *InvalidFileName* exception occurs.
OE0599	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-42, Section 3.1.3.2.3.6.3.5, Page 3-93, Section 3.1.3.6.4
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	C-3 to C-4, C-23

Test Objective

This test case verifies OE0269, OE0270 and OE0599. The objective of this test is to verify that the DomainManager *installApplication* operation raises the *InvalidFileName* exception when the input SAD file or any of the SAD's referenced filenames do not exist in the file system identified by the absolute path of the input *profileFileName* parameter. Also, verify that DomainManager *installApplication* operation logs a FAILURE_ALARM log record to the DomainManager's log with an appropriate detailed message containing the filename that caused the *InvalidFileName* to be raised. Finally, verify the exception contains an error number of the type CF::ErrorNumberType.

Places to Verify

DomainManager

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_EMSGSIZE, 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 InvalidProfile{};
exception ApplicationInstallationError {
        ErrorNumberType errorNumber;
        string msg; };
exception ApplicationAlreadyInstalled{};
```

Operations

```
void installApplication ( in string profileFileName )
    raises ( CF::InvalidProfile, CF::InvalidFileName,
        CF::DomainManager::ApplicationInstallationError,
        CF::DomainManager::ApplicationAlreadyInstalled);
```

Preconditions

- The DomainManager source code files are available.

Test Description

A. Identify the source code files that implement the DomainManager *installApplication* operation. (OE0269, OE0270, OE0599)

   1. **Untested**: The DomainManager *installApplication* operation source code files are not available.

For each DomainManager *installApplication* operation found within the OE under test do the following steps

B. Verify that the DomainManager *installApplication* operation raises the CF::*InvalidFileName* when the input SAD file or any of the SAD's referenced filenames do not exist in the file system identified by the absolute path of the input *profileFileName* parameter. (OE0269).

1. **Pass:** The *installApplication* operation raises the *InvalidFileName* exception when the input SAD file or any of the SAD's referenced filenames do not exist.
2. **Fail:** The *installApplication* operation does not raise the *InvalidFileName* exception when the input SAD file or any of the SAD's referenced filenames do not exist.

C. Verify that a FAILURE_ALARM log record is written to the domain manager's log with a message consisting of "installApplication::invalid file is xxx", where "xxx" is the input or referenced filename, when the *installApplication* operation raises the *InvalidFileName* exception. (OE0270)
   1. **Pass:** A FAILURE_ALARM record is written to the domain manager's log with the above-mentioned circumstances.
   2. **Fail:** No FAILURE_ALARM record is written to the domain manager's log with the above-mentioned circumstances.

D. Verify that the error number that accompanies the exception is a CF ErrorNumberType value. (OE0599)
   1. **Pass:** The *rmdir* exception provides an error number with a CF ErrorNumberType value for the error condition.
   2. **Fail:** The *rmdir* 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_132				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the DomainManager *installApplication* operation. (OE0269, OE0270, OE0599)				
1. Perform a search for *installApplication* operation on all DomainManager source code provided by the developer	**Untested:** No results from search. (OE0269, OE0270, 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 *installApplication* operation implementation. Record the file name.	The filename of the *installApplication* operation for each implementation is identified and recorded.			
For each DomainManager *installApplication* operation found within the OE under test do the following steps				
B. Verify that the DomainManager *installApplication* operation raises the CF::*InvalidFileName* when the input SAD file or any of the SAD's referenced filenames do not exist in the file system identified by the absolute path of the input *profileFileName* parameter. (OE0269)				
3. Search within the *installApplication* operation for where the *CF::InvalidFileName* exception is mentioned.	Identified the source code where *installApplication* operation raises the CF::*InvalidFileName*.			

OE_TC_132				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that *CF::InvalidFileName* is raised when the input SAD file does not exist in the file system identified by the absolute path of the input *profileFileName* parameter.	**Pass:** The *installApplication* operation raises the CF::*InvalidFileName* when the input SAD file does not exist. (OE0269) **Fail:** The *installApplication* operation does not raise the CF::*InvalidFileName* when the input SAD file does not exist. (OE0269)		Note that since there are two reasons for raising the exception: • when the input SAD file does not exist OR • when any of the SAD's referenced filenames do not exist This exception may be raised in more than one place within the *installApplication* operation.	
5. Verify that *CF::InvalidFileName* is raised when any of the SAD's referenced filenames do not exist in the file system identified by the absolute path of the input *profileFileName* parameter.	**Pass:** The *installApplication* operation raises the CF::*InvalidFileName* when any of the SAD's referenced filenames do not exist. (OE0269) **Fail:** The *installApplication* operation does not raise the CF::*InvalidFileName* when any of the SAD's referenced filenames do not exist. (OE0269)			
C. Verify that a FAILURE_ALARM log record is written to the domain manager's log with a message consisting of "installApplication::invalid file is xxx", where "xxx" is the input or referenced filename, when the *installApplication* operation raises the *InvalidFileName* exception. (OE0270)				
6. Verify in the source code that a FAILURE_ALARM log record is written to the domain manager's log when the *InvalidFileName* exception is raised.	**Pass**: A record is written to the domain manager's log. (OE0270) **Fail**: A record is not written to the domain manager's log. (OE0270)			

OE_TC_132				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Verify that log message consists of "installApplication::invalid file is xxx",	**Pass**: The log message is in the proper format (OE0270) **Fail**: The log message is not in the proper format. (OE0270)			
D. Verify that the error number that accompanies the exception is a CF ErrorNumberType value. (OE0599)				
8. Verify if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_132				
Step 2 (Implementation's source file name)	**Step 4 & 5** (Exception raised) Y/N	**Step 6 & 7** (Proper log record written to DMs log) Y/N	**Step 8** (Correct error number type) Y/N	**Notes**

Test Summary OE_TC_132

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):

OE0269_____

OE0270 _____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.29. OE_TC_154 - DomainManager :: registerDeviceManager sends event to ODM

Test Case Number: OE_TC_154

DomainManager::registerDeviceManager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0234	The *registerDeviceManager* operation shall send a DomainManagementObjectAddedEventType event to the Outgoing Domain Management event channel upon successful registration of a device manager.
OE0234-C041	The *producerId* is the identifier attribute of the domain manager.
OE0234-C042	The *sourceId* is the identifier attribute of the registered device manager.
OE0234-C043	The *sourceName* is the label attribute of the registered device manager.
OE0234-C044	The *sourceIOR* is the object reference for the registered device manager.
OE0234-C045	The *sourceCategory* is "DEVICE_MANAGER".

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-36 through 3-37, Section 3.1.3.2.3.6.1 Pages 3-4, Section 3.1.2.3.2.1.6

Test Objective

This test case verifies OE0234 and OE0234-C041 through OE0234-C045. The objective of this test is to verify that the *registerDeviceManager* operation sends a DomainManagementObjectAddedEventType to the Outgoing Domain Management event channel upon successful registration of a device manager. In addition, this test will verify that the DomainManagementObjectAddedEventType event sent by the *registerDeviceManager* operation contains the appropriate data for *producerId*, *sourceId*, *sourceName*, *sourceIOR* and *sourceCategory*.

Places to Verify

DomainManager::registerDeviceManager

IDL References

Data

```
struct DomainManagementObjectAddedEventType {
        string producerId;
        string sourceId;
        string sourceName;
        SourceCategoryType sourceCategory;
        Object sourceIOR;
};
```

Operations

```
void registerDeviceManager (in DeviceManager deviceMgr)
        raises (InvalidObjectReference, InvalidProfile, RegisterError);
```

Preconditions

- The *DomainManager*::*registerDeviceManager* source code files are available.

Test Description

A. Locate the source code for the *DomainManager::registerDeviceManager* operation. (OE0234)
   1. **Pass:** The source code for the *DomainManager::registerDeviceManager* operation is found.
   2. **Untested**: The source code for the *DomainManager::registerDeviceManager* operation in not found.

B. Verify that the *registerDeviceManager* operation sends a DomainManagementObjectAddedEventType event to the Outgoing Domain Management event channel upon successful registration of a device manager. (OE0234)
   1. **Pass:** The *registerDeviceManager* operation sends a DomainManagementObjectAddedEventType event to the Outgoing Domain Management event channel upon successful registration of a device manager.
   2. **Fail:** The *registerDeviceManager* operation does not send a DomainManagementObjectAddedEventType event to the Outgoing Domain Management event channel upon successful registration of a device manager.

C. Verify that the DomainManagementObjectAddedEventType event sent by the *registerDeviceManager* operation contains the appropriate types for *producerId*, *sourceId*, *sourceName*, *sourceIOR* and *sourceCategory*.
   1. Verify that the *producerId* is the identifier attribute of the domain manager. (OE0234-C041)
      a. **Pass:** The *producerId* is the identifier attribute of the domain manager.
      b. **Fail:** The *producerId* is not the identifier attribute of the domain manager.

2. Verify that the *sourceId* is the identifier attribute of the registered device manager. (OE0234-C042)
   a. **Pass:** The *sourceId* is the identifier attribute of the registered device manager.
   b. **Fail:** The *sourceId* is not the identifier attribute of the registered device manager.
3. Verify that the *sourceName* is the label attribute of the registered device manager. (OE0234-C043)
   a. **Pass:** The *sourceName* is the label attribute of the registered device manager.
   b. **Fail:** The *sourceName* is not the label attribute of the registered device manager.
4. Verify that the *sourceIOR* is the object reference for the registered device manager. (OE0234-C044)
   a. **Pass:** The *sourceIOR* is the object reference for the registered device manager.
   b. **Fail:** The *sourceIOR* is not the object reference for the registered device manager.
5. Verify that the *sourceCategory* is "DEVICE_MANAGER". (OE0234-C045)
   a. **Pass:** The *sourceCategory* is "DEVICE_MANAGER".
   b. **Fail:** The *sourceCategory* is not "DEVICE_MANAGER".

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_154				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
For each domain manager, if more than one exists, do the following steps:				
A. Locate the source code for the *DomainManager registerDeviceManager* operation. (OE0234)				
1. Perform a search on the source code for the implementation of the *DomainManager registerDeviceManager* operation and record the source code file name.	**Pass:** The source code for the *DomainManager registerDeviceManager* operation is found. (OE0234) **Untested**: The source code for the *DomainManager registerDeviceManager* operation in not found. (OE0234)			
B. Verify that the *registerDeviceManager* operation sends a DomainManagementObjectAddedEventType event to the Outgoing Domain Management event channel upon successful registration of a device manager. (OE0234)				
2. Verify that the *registerDeviceManager* operation sends a DomainManagementObjectAddedEventType event to the Outgoing Domain Management event channel upon successful registration of a device manager.	**Pass:** The *registerDeviceManager* operation sends a DomainManagementObjectAddedEventType event to the Outgoing Domain Management event channel upon successful registration of a device manager. (OE0234) **Fail:** The *registerDeviceManager* operation does not send a DomainManagementObjectAddedEventType event to the Outgoing Domain Management event channel upon successful registration of a device manager. (OE0234)		Default Incoming and Outgoing Domain Management channels are created by the Domain Manager.	
C. Verify that the DomainManagementObjectAddedEventType event sent by the *registerDeviceManager* operation contains the appropriate data for *producerId, sourceId, sourceName, sourceIOR* and *sourceCategory*.				

OE_TC_154				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *producerId* is the identifier attribute of the domain manager.	**Pass:** The *producerId* is the identifier attribute of the domain manager. (OE0234-C041) **Fail:** The *producerId* is not the identifier attribute of the domain manager. (OE0234-C041)		Failure of this criterion OE0234-C041 means failure of the requirement OE0234.	
4. Verify that the *sourceId* is the identifier attribute of the registered device manager.	**Pass:** The *sourceId* is the identifier attribute of the registered device manager. (OE0234-C042) **Fail:** The *sourceId* is not the identifier attribute of the registered device manager. (OE0234-C042)		Failure of this criterion OE0234-C042 means failure of the requirement OE0234.	
5. Verify that the *sourceName* is the label attribute of the registered device manager.	**Pass:** The *sourceName* is the label attribute of the registered device manager. (OE0234-C043) **Fail:** The *sourceName* is not the label attribute of the registered device manager. (OE0234-C043)		Failure of this criterion OE0234-C043 means failure of the requirement OE0234.	
6. Verify that the *sourceIOR* is the object reference for the registered device manager.	**Pass:** The *sourceIOR* is the object reference for the registered device manager. (OE0234-C044) **Fail:** The *sourceIOR* is not the object reference for the registered device manager. (OE0234-C044)		Failure of this criterion OE0234-C044 means failure of the requirement OE0234.	
7. Verify that the *sourceCategory* is "DEVICE_MANAGER".	**Pass:** The *sourceCategory* is "DEVICE_MANAGER". (OE0234-C045) **Fail:** The *sourceCategory* is not "DEVICE_MANAGER". (OE0234-C045)		Failure of this criterion OE0234-C045 means failure of the requirement OE0234.	
End of Test				

Test Recording Log – OE_TC_154						
Step 1 (source code file name)	Step 2 (sends event) Y/N	Step 3 (domain manager identifier) Y/N	Step 4 (device manager identifier) Y/N	Step 5 (device manager label) Y/N	Step 6 (device manager object reference) Y/N	Step 7 (DEVICE_ MANAGER) Y/N

Test Summary OE_TC_154

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):

OE0234_____

OE0234-C041 _____

OE0234-C042 _____

OE0234-C043 _____

OE0234-C044 _____

OE0234-C045 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.30. OE_TC_157 - DomainManager :: registerDeviceManager establishes connections

Test Case Number: OE_TC_157

DomainManager::registerDeviceManager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0226	The *registerDeviceManager* operation shall establish any connections for the device manager indicated by the input deviceMgr parameter, which are specified in the *connections* element of the device manager's Device Configuration Descriptor (DCD) file, that are possible with the current set of registered devices and services.
OE0226-C040	Connections not currently possible are left unconnected pending future device / service registrations.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-37, Section 3.1.3.2.3.6.1.3;

Test Objective

This test case verifies requirement OE0226 and criterion OE0226-C040. The objective of this test is to verify that the *registerDeviceManager* operation establishes connections for the device manager in the input deviceMgr parameter. The device manager's DCD file specifies the connections for the DeviceManager. Also, verify that if the connection is not currently possible then the connection is left unconnected with no error.

Places to Verify

DomainManager

IDL References

Operations

void *registerDeviceManager* (in CF::DeviceManager deviceMgr)
raises (CF::InvalidObjectReference,CF::InvalidProfile, CF::DomainManager::RegisterError);

Preconditions

- Each DeviceManager has a corresponding DCD file

The *registerDeviceManager* operation implementation(s) source code file(s) are available.

Test Description

Note: In the referenced SCA Specification v2.2.2, there is a Figure 3-15 which contains a sequence diagram for the processing associated with the *registerDeviceManager* operation. The requirement being tested here, OE0226, is addressed in the optional steps 11 through 14 and 15 through 18 in the diagram. Steps 11 through 14 are optional for each Device, provided the Device requires a service and the service exists in the domain. Steps 15 through 18 are optional, provided the device manager is using a service and the service exists in the domain.

For each DomainManager within the OE under test, perform the following steps:

A. Locate the source code for the DomainManager *registerDeviceManager* operation. (OE0226)
   1. **Pass**: The source code for the DomainManager *registerDeviceManager* operation is found.
   2. **Untested**: The source code for the DomainManager *registerDeviceManager* operation is not found.

B. Verify that the *registerDeviceManager* operation uses the connections specified in the device manager's DCD file, that are possible with the current set of registered devices and services. (OE0226)
   1. **Pass:** The *registerDeviceManager* operation uses the connections specified in the device manager's DCD file that are possible with the current set of registered devices and services.
   2. **Fail:** The *registerDeviceManager* operation does not use the connections specified in the device manager's DCD file that are possible with the current set of registered devices and services.

For each instance of a connection that is possible with the current set of registered devices and services, perform the following steps:

C. Verify that the *registerDeviceManager* operation establishes the connection. (OE0226)
   1. **Pass:** The *registerDeviceManager* operation establishes the connection. (Skip step D and look at the next possible connection).

D. If the previous step does not Pass, verify that the *registerDeviceManager* operation leaves the connection unconnected (pending future device / service registrations). (OE0226, OE0226-C040)
   1. **Pass:** The *registerDeviceManager* operation leaves the connection unconnected and performs no error processing.
   2. **Fail:** The *registerDeviceManager* operation does not leave the connection unconnected or performs some error processing.

Semi-automated Test Steps

After running JTAP's DomainManager registerDeviceManager unregisterDeviceManager test case, verify in the log file that possible connections are made (OE0226), perform manual steps 1-4 to test the criterion OE0226-C040.

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_157				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
For each DomainManager within the OE under test				
A. Locate the source code for the *DomainManager registerDeviceManager* operation. (OE0226)				
1. Perform a search on all source code provided by the developer for the implementation of the *DomainManager registerDeviceManager* operation and record the source code file name.	**Pass:** The source code for the *DomainManager registerDeviceManager* operation is found. (OE0226) **Untested**: The source code for the *DomainManager registerDeviceManager* operation is not found. (OE0226)			
B. Verify that the *registerDeviceManager* operation uses the connections specified in the device manager's DCD file that are possible with the current set of registered devices and services. (OE0226)				
2. Verify within the *registerDeviceManager* operation that the operation uses a collection of device and service connections derived from the input *deviceMgr* parameter object's DCD files.	**Pass:** The *registerDeviceManager* operation uses a collection of device and service connections derived from the input *deviceMgr* parameter object's DCD files. (OE0226) **Fail:** The *registerDeviceManager* operation does not use a collection of device and service connections derived from the input *deviceMgr* parameter object's DCD files. (OE0226)		The code should use the input object to get a list or collection of connections. It would then loop though the connections, acting on each one. Connects that are possible are tested in Step 3 below. Connections that are not possible are tested in Step 4 below.	

OE_TC_157				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
For each instance of a connection that is possible with the current set of registered devices and services, perform the following steps:				
C. Verify that the *registerDeviceManager* operation establishes the connection. (OE0226)				
3. Verify within the *registerDeviceManager* operation that the connection is established.	**Pass:** The *registerDeviceManager* operation establishes the connection. (OE0226)		For a given connection candidate, if this step passes, then skip step 4 and look at the next connection candidate.	
D. If the previous step does not Pass, verify that the *registerDeviceManager* operation leaves the connection unconnected pending future device / service registrations. (OE226, OE0226-C040)				
4. Verify within the *registerDeviceManager* operation that the connection is left unconnected (pending future device/service registrations), and without error.	**Pass:** If no connection is possible, the *registerDeviceManager* operation leaves the connection unconnected and without error. (OE0226-C040) **Fail:** If no connection is possible, the *registerDeviceManager* operation does not leave the connection unconnected and without error. (OE0226, OE0226-C040)		The portion of the criteria that states "pending future device/service registrations" is not testable since it assumes future behavior of the system. Failure of this criterion OE0226-C040 means failure of the requirement OE0226.	
End of Test				

Test Recording Log – OE_TC_157			
Step1 (locate registerDeviceManager source code)	**Step2** (use deviceMgr to get list of connections)	**Step3** (connect to those that are possible)	**Step4** (don't connect to those that are not possible)

Test Summary OE_TC_157

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):

OE0226_____

OE0226-C040 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.31. OE_TC_158 - DomainManager :: unregisterDeviceManager

Test Case Number: OE_TC_158
DomainManager::unregisterDeviceManager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0279	The *unregisterDeviceManager* operation shall send a DomainManagementObjectRemovedEventType event to the Outgoing Domain Management event channel, upon successful unregistration of a device manager.
OE0279-C059	The *producerId* is the identifier attribute of the domain manager.
OE0279-C060	The *sourceId* is the identifier attribute of the unregistered device manager.
OE0279-C061	The *sourceName* is the label attribute of the unregistered device manager.
OE0279-C062	The *sourceCategory* is "DEVICE_MANAGER".

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-43 through 3-44, Section 3.1.3.2.3.6.4 Pages 3-4, Section 3.1.2.3.2.1.5

Test Objective

This test case verifies OE0279 and OE0279-C059 through OE0279-C062. The objective of this test is to verify that the *unregisterDeviceManager* operation sends a DomainManagementObjectRemovedEventType to the Outgoing Domain Management event channel upon successful unregistration of a device manager. In addition, this test will verify that the DomainManagementObjectRemovedEventType event sent by the *unregisterDeviceManager* operation contains the appropriate data for *producerId, sourceId, sourceName* and *sourceCategory*.

Places to Verify

DomainManager unregisterDeviceManager

IDL References

Data

```
struct DomainManagementObjectRemovedEventType{
        string producerId; string sourceId; string sourceName; SourceCategoryType sourceCategory;
```

};

Operations

void unregisterDeviceManager (in DeviceManager deviceMgr)
raises (InvalidObjectReference, UnregisterError);

Preconditions

- The *DomainManager* source code files are available.

Test Description

A. Locate the source code for the *DomainManager unregisterDeviceManager* operation. (OE0279)
   1. **Pass:** The source code for the *DomainManager unregisterDeviceManager* operation is found.
   2. **Untested**: The source code for the *DomainManager unregisterDeviceManager* operation in not found.

B. Verify that the *unregisterDeviceManager* operation sends a DomainManagementObjectRemovedEventType event to the Outgoing Domain Management event channel upon successful unregistration of a device manager. (OE0279)
   1. **Pass:** The *unregisterDeviceManager* operation sends a DomainManagementObjectRemovedEventType event to the Outgoing Domain Management event channel upon successful unregistration of a device manager.
   2. **Fail:** The *unregisterDeviceManager* operation does not send a DomainManagementObjectRemovedEventType event to the Outgoing Domain Management event channel upon successful unregistration of a device manager.

C. Verify that the DomainManagementObjectRemovedEventType event sent by the *unregisterDeviceManager* operation contains the appropriate data for *producerId*, *sourceId*, *sourceName* and *sourceCategory*.
   1. Verify that the *producerId* is the identifier attribute of the domain manager. (OE0279-C059)
      a. **Pass:** The *producerId* **is** the identifier attribute of the domain manager.
      b. **Fail:** The *producerId* is **not** the identifier attribute of the domain manager.
   2. Verify that the *sourceId* is the identifier attribute of the unregistered device manager. (OE0279-C060)
      a. **Pass:** The *sourceId* **is** the identifier attribute of the unregistered device manager.
      b. **Fail:** The *sourceId* is **not** the identifier attribute of the unregistered device manager.
   3. Verify that the *sourceName* is the label attribute of the unregistered device manager. (OE0279-C061)
      a. **Pass:** The *sourceName* **is** the label attribute of the unregistered device manager.
      b. **Fail:** The *sourceName* is **not** the label attribute of the unregistered device manager.
   4. Verify that the *sourceCategory* is "DEVICE_MANAGER". (OE0279-C062)
      a. **Pass:** The *sourceCategory* **is** "DEVICE_MANAGER".
      b. **Fail:** The *sourceCategory* is **not** "DEVICE_MANAGER".

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_158				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the *DomainManager unregisterDeviceManager* operation. (OE0279)				
1. Perform a search on all source code provided by the developer for the implementation of the *DomainManager unregisterDeviceManager* operation and record the source code file name.	**Pass:** The source code for the *DomainManager unregisterDeviceManager* operation is found. (OE0279) **Untested**: The source code for the *DomainManager unregisterDeviceManager* operation in not found. (OE0279)			
B. Verify that the *unregisterDeviceManager* operation sends a DomainManagementObjectRemovedEventType event to the Outgoing Domain Management event channel upon successful unregistration of a device manager. (OE0279)				
2. From the source code file recorded in the previous step Verify that the *unregisterDeviceManager* operation sends a DomainManagementObjectRemovedEventType event to the Outgoing Domain Management event channel upon successful unregistration of a device manager.	**Pass:** The *unregisterDeviceManager* operation sends a DomainManagementObjectRemovedEventType event to the Outgoing Domain Management event channel upon successful unregistration of a device manager. (OE0279) **Fail:** The *unregisterDeviceManager* operation does not send a DomainManagementObjectRemovedEventType event to the Outgoing Domain Management event channel upon successful unregistration of a device manager. (OE0279)		Default Incoming and Outgoing Domain Management channels are created by the Domain Manager.	
C. Verify that the DomainManagementObjectRemovedEventType event sent by the *unregisterDeviceManager* operation contains the appropriate data for *producerId, sourceId, sourceName* and *sourceCategory*.				
1. Verify that the *producerId* is the identifier attribute of the domain manager. (OE0279-C059)				

OE_TC_158				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *producerId* is the identifier attribute of the domain manager.	**Pass:** The *producerId* **is** the identifier attribute of the domain manager. (OE0279-C059) **Fail:** The *producerId* is **not** the identifier attribute of the domain manager. (OE0279-C059)		Failure of this criterion OE0279-C059 means failure of the requirement OE0279.	
2. Verify that the *sourceId* is the identifier attribute of the unregistered device manager. (OE0279-C060)				
4. Verify that the *sourceId* is the identifier attribute of the unregistered device manager.	**Pass:** The *sourceId* **is** the identifier attribute of the unregistered device manager. (OE0279-C060) **Fail:** The *sourceId* is **not** the identifier attribute of the unregistered device manager. (OE0279-C060)		Failure of this criterion OE0279-C060 means failure of the requirement OE0279.	
3. Verify that the *sourceName* is the label attribute of the unregistered device manager. (OE0279-C061)				
5. Verify that the *sourceName* is the label attribute of the unregistered device manager.	**Pass:** The *sourceName* **is** the label attribute of the unregistered device manager. (OE0279-C061) **Fail:** The *sourceName* is **not** the label attribute of the unregistered device manager. (OE0279-C061)		Failure of this criterion OE0279-C061 means failure of the requirement OE0279.	
4. Verify that the *sourceCategory* is "DEVICE_MANAGER". (OE0279-C062)				
6. Verify that the *sourceCategory* is "DEVICE_MANAGER".	**Pass:** The *sourceCategory* **is** "DEVICE_MANAGER". (OE0279-C062) **Fail:** The *sourceCategory* is **not** "DEVICE_MANAGER". (OE0279-C062)		Failure of this criterion OE0279-C062 means failure of the requirement OE0279.	
End of Test				

Test Recording Log – OE_TC_158					
Step1 (source code file name)	**Step2** (sends event) Y/N	**Step3** (domain manager identifier) Y/N	**Step4** (device manager identifier) Y/N	**Step5** (device manager label) Y/N	**Step6** (DEVICE_ MANAGER) Y/N

Test Summary OE_TC_158

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):

OE0279_____

OE0279-C059 _____

OE0279-C060 _____

OE0279-C061 _____

OE0279-C062 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.32. OE_TC_162 - ApplicationFactory :: create deallocates capacity on devices

Test Case Number: OE_TC_162

ApplicationFactory::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0708	The *create* operation shall deallocate any capacity allocations on devices that do not satisfy the application components allocation requirements or that are not utilized due to an unsuccessful application creation.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-27 through 3-29, Section 3.1.3.2.2.5.1.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-2, C-23

Test Objective

This test case verifies OE0708. The objective of this test is to verify that the *create* operation shall deallocate any capacity allocations on devices that do not satisfy the application components allocation requirements or that are not utilized due to an unsuccessful application creation.

Places to Verify

CF::Application create

IDL References

Data

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

Operations

```
CF::Application create
        ( in string name,
```

```
        in CF::Properties initConfiguration,
        in CF::DeviceAssignmentSequence deviceAssignments )
raises ( CF::ApplicationFactory::CreateApplicationError,
        CF::ApplicationFactory::CreateApplicationRequestError,
        CF::ApplicationFactory::InvalidInitConfiguration);

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

Preconditions

- All the ApplicationFactory source code files are available.

Test Description

A. Locate all implementations of CF::Application *create* operation. (OE0708)
   1. **Pass**: Implementations of CF::Application *create* are found.
   2. **Fail**: No implementations of CF::Application *create* are found.

B. Verify that the capacity allocations are deallocated if that device cannot meet application components allocation requirements. (OE0708)
   1. **Pass:** The capacity allocations are deallocated if the device fails to meet the application components application requirements.
   2. **Fail:** The capacity allocations are not deallocated if the device fails to meet the application components application requirements.

C. Verify that the capacity allocations are deallocated if the creation of an application fails. (OE0708)
   1. **Pass**: Capacity allocations are deallocated if creation of an application fails.
   2. **Fail**: Capacity allocations are not deallocated if creation of an application fails.

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_162				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all implementations of CF::Application *create* operation. (OE0708)				
1. Perform a search of the source code for all implementations of CF::Application *create* and record the file name(s).	**Pass**: Implementations of CF::Application *create* are found. (OE0708) **Fail**: No implementations of CF::Application *create* are found. (OE0708)		This should never FAIL since the *create* operation is the only way to instantiate an application on a radio. CF::Application* *CF__ApplicationFactory_impl::create* (const CORBA::Char* name, const CF::Properties& InitConfiguration, const CF::DeviceAssignmentSequence& DeviceAssignments, CORBA::Environment&_env	
B. Verify that the capacity allocations are deallocated if that device cannot meet the application components application requirements. (OE0708)				
2. Verify that the capacity allocations are deallocated if that device cannot meet the application components application requirements.	**Pass:** The capacity allocations are deallocated if that device cannot meet the application components application requirements. (OE0708) **Fail:** The capacity allocations are not deallocated if that device cannot meet the application components application requirements. (OE0708)		Look for **CreateApplicationRequestError** exception. It is raised when the input CF DeviceAssignmentSequence parameter contains one or more invalid application component to device assignments. When this exception is raised, the application will clean up.	
C. Verify that the capacity allocations are deallocated if the creation of an application fails. OE0708)				

OE_TC_162				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that capacity allocations are deallocated if the applications create fails.	**Pass**: Capacity allocations are deallocated if the applications create fails. (OE0708) **Fail**: Capacity allocations are not deallocated if the applications create fails. (OE0708		Each assignment forces a component instantiation to a specified device. Look for the **CreateApplicationError** exception. This occurs when the *create* request is valid but the application cannot be successfully instantiated due to internal processing error(s).	
End of Test				

Test Recording Log – OE_TC_162		
Step 1 (CF::Application *create*() file names)	**Step 2** (CreateApplicationRequestError exception is raised – Y/N?)	**Step 3** (CreateApplicationError exception is raised – Y/N?)

Test Summary OE_TC_162

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):

OE0708_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.33. OE_TC_164 - ApplicationFactory :: create establishes connections for named applications

Test Case Number: OE_TC_164

CF::ApplicationFactory::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0710	The *create* operation shall establish connections for an application which are specified in the SAD *domainfinder* element.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-27 through 3-31, Section 3.1.3.2.2.5.1.3
SCA Appendix C – IDL	Version 2.2.2 15 May 2006	Pages C-2, C-22 and C-23
SCA Appendix D – Domain Profile	Version 2.2.2 15 May 2006	Pages D-40, Section D.6.1.5.1 Pages D-43, Section D.6.1.5.1.1.5

Test Objective

This test case verifies OE0710. The objective of this test is to verify that Operating Environment(OE) provides the functionality in the CF::ApplicationFactory *create* operation to establish all of the connections for an application. The connections for the application being instantiated are specified by the application's SAD file, specified in the domainfinder element.

Places to Verify

CF::ApplicationFactory create

IDL References

Data

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

Operations

```
CF::Application create
        ( in string name,
          in CF::Properties initConfiguration,
          in CF::DeviceAssignmentSequence deviceAssignments )
raises (
        CF::ApplicationFactory::CreateApplicationError,
        CF::ApplicationFactory::CreateApplicationRequestError,
        CF::ApplicationFactory::InvalidInitConfiguration);
```

Preconditions

- All the ApplicationFactory source code files are available.

Test Description

A. Verify the CF::ApplicationFactory *create* operation will establish connections specified by an application's SAD file. (OE0710)
   1. **Pass:** The *create* operation establishes connections specified in the application's SAD file.
   2. **Fail:** The *create* operation does not establish connections specified in the application's SAD 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_164				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify the CF::ApplicationFactory *create* operation will establish connections specified by an application's SAD file. (OE0710				
1. Locate and record the file names where all instances of CF::ApplicationFactory *create* are implemented.	The *create* operation is located.			
2. Locate in the CF::ApplicationFactory *create* source code where information from the SAD files domainfinder is processed.	**Pass**: Source code where the SAD files domainfinder is processed is found. (OE0710) **Fail**: Source code where the SAD files domainfinder is processed is not found. (OE0710)			
3. Verify that all the connections that can be specified in an application's SAD file are established.	**Pass**: All connections that can be specified in an application's SAD file are established. (OE0710) **Fail**: Not all connections that can be specified in an application's SAD file are established. (OE0710)			
End of Test				

Test Recording Log – OE_TC_164		
Step 1 (CF::ApplicationFactory create() file name)	**Step 2** (connections established: Y/N)	**Notes**

Test Summary OE_TC_164

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):

OE0710_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.34. OE_TC_165 - ApplicationFactory :: create defines an order for initialization

Test Case Number: OE_TC_165

ApplicationFactory::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0174	The *create* operation shall, in order, initialize all application resources, then establish connections for those resources, and finally configure the application component indicated by the assemblycontroller element in the SAD.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-15, Figure 3-8; Pages 3-27 through 3-29, Section 3.1.3.2.2.5.1.3
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Pages D-33 thru D-34, Section D.6.1; Page D-40, Section D.6.1.4

Test Objective

This test case verifies OE0174. The objective of this test is to verify the order in which the *create* operation will perform some of its tasks. Specifically it requires all resources to be initialized, then all connections to be made, and finally, the component to be configured. The assembly controller is always required because all external requests of runTest, start, stop, configure, and query are delegated through it to the proper application.

Places to Verify

ApplicationFactory::create

IDL References

Operations

```
Application create (in string name, in Properties initConfiguration, in DeviceAssignmentSequence deviceAssignments)
    raises (CreateApplicationError, CreateApplicationRequestError, InvalidInitConfiguration);
```

Preconditions

- The CF source code files are available.

Test Description

A. Locate all occurrences of the implementation of the *CF::ApplicationFactory::create()* operation. (OE0174)
   1. **Untested**: No implementations of the *CF::ApplicationFactory::create()* operation can be found.

For each instantiation of the *CF::ApplicationFactory::create()* operation perform the following:

B. Locate the initialization of application resources.
C. Verify that all application resources are initialized before any connections are attempted. (OE0174).
   1. **Pass:** All application resources are initialized before any connections are attempted.
   2. **Fail:** One or more application resources are not initialized before any connections are attempted.
D. Verify that all connections are completed before the application component is configured. (OE0174).
   1. **Pass:** All connections are completed before the application component is configured.
   2. **Fail:** One or more connections are not completed before the application component is configured.

Semi-automated Test Steps

After running JTAP's ApplicationFactory create PseudoWaveform test, perform manual steps 1 through 6.

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_165				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all occurrences of the implementation of the *CF::ApplicationFactory::create()* operation. (OE0174)				
1. Search the source code for all occurrences of the *CF::ApplicationFactory::Create operation* and record the files in which it is found.	**Untested**: the *create* operation cannot be found. (OE0174)			
For each instantiation of the *CF::ApplicationFactory::create()* operation perform the following:				
B. Locate the initialization of application resources.				
2. Locate where the *create* operation initializes the application resources (invokes the *Lifecycle::initialize* operation).	The invocation of the *initialize* operation is found.			
C. Verify that all application resources are initialized before any connections are attempted. (OE0174).				
3. Verify that the *initialize* operation is invoked for all resources before any *port::connectPort* invocations occur.	**Pass**: No *connectPort* operations are invoked before the *initialize* operation is invoked. (OE0174) **Fail**: One or more *connectPort* operations are invoked before the *initialize* operation is invoked. (OE0174)			
D. Verify that all connections are completed before the application component is configured. (OE0174).				

OE_TC_165				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Locate the *port::connectPort* invocations for the resources.	The invocation of the *connectPort* operation is found.			
5. Locate where the *assemblycontroller* element in the SAD is processed and determine if the *componentinstantiationref* is saved.	**Pass**: The *componentinstantiationref* is saved. (OE0174) **Fail**: The *componentinstantiationref* is not saved. (OE0174)		The *assemblycontroller* element is required and it requires a *componentinstantiationref*.	
6. Verify that all the *connectPort* operations are invoked before the *PropertySet::configure* operation of the component identified in the SAD's *assembly controller* element is invoked.	**Pass**: All *connectPort* operations are invoked before the *configure* operation is invoked. (OE0174) **Fail**: One or more *connectPort* operations are invoked after the *configure* operation is invoked. (OE0174)		The *componentinstantiationref* points to the object whose *configure* operation will be invoked.	
End of Test				

Test Recording Log – OE_TC_165			
Step 1 (ApplicationFactory::create files)	**Step 3** (initialization first)	**Step 5** (*componentinstantiationref*)	**Step 6** (*connectPort* before configuration)

Test Summary OE_TC_165

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):

OE0174_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.35. OE_TC_166 - DomainManager :: registerDevice verifies input parameters

Test Case Number: OE_TC_166

DomainManager::registerDevice

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0715	The *registerDevice* operation shall verify that the input parameters, registeringDevice and registeredDeviceMgr, are not nil CORBA object references.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-39, Section 3.1.3.2.3.6.2.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-19, Section C.1

Test Objective

This test case verifies OE0715. The objective of this test is to verify that the *registerDevice* operation validates its parameters (registeringDevice and registeredDeviceMgr) are not nil CORBA object references.

Places to Verify

DomainManager registerDevice

IDL References

Operations

void *registerDevice* (in CF::Device registeringDevice, in CF::DeviceManager registeredDeviceMgr)
raises (CF::InvalidObjectReference, CF::InvalidProfile, CF::DomainManager::DeviceManagerNotRegistered, CF::DomainManager::RegisterError);

Preconditions

- The *DomainManager registerDevice* source code files are available.

Test Description

A. Locate the source code for the *DomainManager registerDevice* operation. (OE0715)
   1. **Pass:** The source code for the *DomainManager registerDevice* operation is found.
   2. **Failed:** The source code for the *DomainManager registerDevice* operation in not found.

B. Confirm that the *registerDevice* operation verifies that the CF::Device registeringDevice parameter is not a nil CORBA object reference. (OE0715)
   1. **Pass:** The *registerDevice* operation verifies that the parameter CF::Device registeringDevice is not a nil CORBA object reference.
   2. **Fail:** The *registerDevice* operation does not confirm that the parameter CF::Device registeringDevice is not a nil CORBA object reference.

C. Confirm that the *registerDevice* operation verifies that the CF::DeviceManager registeredDeviceMgr is not a nil CORBA object reference. (OE0715)
   1. **Pass:** The *registerDevice* operation verifies that the parameter CF::DeviceManager registeredDeviceMgr is not a nil CORBA object reference.
   2. **Fail:** The *registerDevice* operation does not confirm that the parameter CF::DeviceManager registeredDeviceMgr is not 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_166				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the *DomainManager registerDevice* operation. (OE0715)				
1. Perform a search on source code for the implementation of the *DomainManager registerDevice* operation and record the source code file name.	**Pass:** The source code for the *DomainManager registerDevice* operation is found. (OE0715) **FAIL:** The source code for the *DomainManager registerDevice* operation in not found. (OE0715)			
B. Confirm that the *registerDevice* operation verifies that the CF::Device registeringDevice parameter is not a nil CORBA object reference. (OE0715)				

2. Confirm that the *registerDevice* operation verifies that the CF::Device registeringDevice parameter is not a nil CORBA object reference.	**Pass:** The *registerDevice* operation confirms that the parameter CF::Device registeringDevice is not a nil CORBA object reference. (OE0715)
	
	Fail: The *registerDevice* operation does not confirm that the parameter CF::Device registeringDevice is not a nil CORBA object reference. (OE0715)
			If(CORBA::is_nil(*registeringDevice*))
			{
			...
			}	

C. Confirm that the *registerDevice* operation verifies that the CF::DeviceManager registeredDeviceMgr parameter is not a nil CORBA object reference. (OE0715)				

3. Confirm that the *registerDevice* operation verifies that the CF::Device registeredDeviceMgr parameter is not a nil CORBA object reference.	**Pass:** The *registerDevice* operation confirms that the parameter CF::Device registeredDeviceMgr is not a nil CORBA object reference. (OE0715)
	
	Fail: The *registerDevice* operation does not confirm that the parameter CF::Device
			If(CORBA::is_nil(*registeredDeviceMgr*)) {
			...
			}	

OE_TC_166				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
	registeredDeviceMgr is not a nil CORBA object reference. (OE0715)			
End of Test				

Test Recording Log – OE_TC_166		
Step 1 (source code file name)	Step 2 (registeringDevice checked for nil) Y/N	Step 3 (registeredDeviceMgr checked for nil) Y/N

Test Summary OE_TC_166

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):

OE0715_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.36. OE_TC_167 - DomainManager :: registerDevice returns without error if device exists

Test Case Number: OE_TC_167

DomainManager::registerDevice

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0716	The *registerDevice* operation shall return without exception and not register a new device when that device, indicated by the input registeringDevice parameter, has the same identifier attribute as a previously registered device and the reference to the registered device refers to an existing object.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-39 through 3-41, Section 3.1.3.2.3.6.2
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-19

Test Objective

This test case verifies OE0716. The objective of this test is to verify that the *registerDevice* operation will not throw an exception and not register a new device, when that device is a duplicate and the reference refers to an existing object. The *registerDevice* operation registers a Device for a specific device manager with the DomainManager.

Places to Verify

DomainManager

IDL References

Operations

```
void registerDevice (
        in CF::Device registeringDevice,
        in CF::DeviceManager registeredDeviceMgr
        )
        raises ( CF::InvalidObjectReference, CF::InvalidProfile,
                CF::DomainManager::DeviceManagerNotRegistered,
                CF::DomainManager::RegisterError);
```

Preconditions

- The DomainManager source code files are available.

Test Description

A. Identify the source code files that implement the DomainManager *registerDevice* operation. (OE0716)
   1. **Pass:** The source code for the *registerDevice* operation is found.
   2. **Untested**: The source code for the *registerDevice* operation is not found.

B. Verify that the *registerDevice* operation will not throw an exception and not register a new device when that device, indicated by the input *registeringDevice* parameter, has the same identifier attribute as a previously registered device and the reference to the registered device refers to an existing object. (OE0716)
   1. **Pass:** The registerDevice operation does not throw an exception and does not a register a new device.
   2. **Fail:** The registerDevice operation throws an exception and/or registers a new 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_167				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the registerDevice operation. (OE0716)				
1. Perform a keyword search on the source code provided by the developer for the implementation of the *registerDevice* operation. Record the source file name.	**Pass:** The *registerDevice* operation exists in the source code. (OE0716) **Untested**: The *registerDevice* operation does not exist in the source code. (OE0716)		An Integrated Development Environment (IDE) is the best tool for this search. Example: void CF__DomainManager_impl::**registerDevice** (	
B. Verify that the registerDevice operation will not throw an exception and not register a new device when that device, indicated by the input registeringDevice parameter, has the same identifier attribute as a previously registered device and the reference to the registered device refers to an existing object. (OE0716)				
2. Locate where the *registerDevice* operation uses the input registeringDevice parameter to register a Device.	The input *registeringDevice* parameter is located.		Example: **registeringDeviceID** = **registeringDevice->identifier**(CF_ENV_SINGLE_ARG_PARAMETER);	

3. Verify that the code is checking for the existence of object reference.	The code checks that the object reference is valid.		Example:
			
			if (CORBA::**is_nil(registeringDevice)** || !passNarrowTest)
			{	

OE_TC_167				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the *registerDevice* operation will not throw an exception and does not register a new device when that device, indicated by the input *registeringDevice* parameter, has the same identifier attribute as a previously registered device and the reference to the registered device refers to an existing object.	**Pass:** The registerDevice operation does not throw an exception and does not a register a new device. (OE0716) **Fail:** The registerDevice operation throws an exception and/or registers a new device. (OE0716)		*check if device is already registered:* CF_Mutex::Guard registerDeviceGuard (_deviceMutex); DeviceMap::iterator curDevice = _theRegisteredDevices.find(**registeringDeviceID**); *check for duplicate; if not add registered Device* _theRegisteredDevices[**registeringDeviceID**] = CF::Device::_duplicate(registeringDevice); *SPD implementation id first* CORBA::String_var implementationID = getComponentImplementationId(registeringDeviceID.in());	
End of Test				

Test Recording Log – OE_TC_167	
Step 1 (source code file name)	**Step 4** (Is an Exception thrown - Y/N?)

Test Summary OE_TC_167

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):

OE0716_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.37. OE_TC_168 - DomainManager :: registerDevice registers if device doesn't exist

Test Case Number: OE_TC_168

DomainManager::registerDevice

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0717	The *registerDevice* operation shall register the new device indicated by the input registeringDevice parameter, when the previously registered device has the same identifier attribute as the new device and the reference to the registered device refers to a nonexistent object.
OE0718	The *registerDevice* operation shall write an ADMINISTRATIVE_EVENT log record when reference to the registered device refers to a nonexistent object.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-39 through 3-41, Section 3.1.3.2.3.6.2
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-19

Test Objective

This test case verifies OE0717 and OE0718. The registerDevice operation should implement a check for registering a device with a previously registered identifier. The registerDevice operation should register the device indicated if the identifier attribute has not been previously registered or if it has been previously registered and the reference is to a nonexistent object. The test also verifies that an ADMINISTRATIVE_EVENT log record is written to the log when the reference to the registered device refers to a nonexistent object.

Places to Verify

Domain Manager

IDL References

Operations

```
void registerDevice (
  in CF::Device registeringDevice,
  in CF::DeviceManager registeredDeviceMgr)
```

raises (CF::InvalidObjectReference, CF::InvalidProfile,
CF::DomainManager::DeviceManagerNotRegistered,
CF::DomainManager::RegisterError);

Preconditions

- The DomainManager source code files are available.

Test Description

A. Locate the source code for the *DomainManager registerDevice* operation. (OE0717, OE0718)
   1. **Pass:** The source code for the *registerDevice* operation is found.
   2. **Untested**: The source code for the *registerDevice* operation is not found.

B. Verify the DomainManager *registerDevice* operation registers the new device indicated by the input registerDevice parameter, when the previously registered device has the same identifier attribute as the new device and the reference to the registered device refers to a nonexistent object. (OE0717)
   1. **Pass:** The *registerDevice* operation registers the new device indicated by the input registerDevice parameter, when the previously registered device has the same identifier attribute as the new device and the reference to the registered device refers to a nonexistent object.
   2. **Fail:** The *registerDevice* operation does not register the new device indicated by the input registerDevice parameter, when the previously registered device has the same identifier attribute as the new device and the reference to the registered device refers to a nonexistent object.
   3. **Fail:** The *registerDevice* operation registers the new device indicated by the input registerDevice parameter, when the previously registered device has the same identifier attribute as the new device and the reference to the registered device refers to an existent object.

C. Verify that the DomainManager *registerDevice* operation writes an ADMINISTRATIVE_EVENT log record when reference to the registered device refers to a nonexistent object. (OE0718)
   1. **Pass:** The *registerDevice* operation writes an ADMINISTRATIVE_EVENT log record when reference to the registered device refers to a nonexistent object.
   2. **Fail:** The *registerDevice* operation does not write an ADMINISTRATIVE_EVENT log record when reference to the registered device refers to a nonexistent object.

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_168				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the *DomainManager registerDevice* operation. (OE0717, OE0718)				
1. Perform a search on all source code provided by the developer for the implementation of the *DomainManager registerDevice* operation and record the source code file name.	**Pass:** The source code for the *DomainManager registerDevice* operation is found. (OE0717) **Untested**: The source code for the *DomainManager registerDevice* operation is not found. (OE0717)		An Integrated Development Environment (IDE) is the best tool for this search. The operation may look like this: void DeviceManager_Servant::**registerDevice** (CF::Device_ptrregisteringDevice CF_ENV_ARG_DECL)	
B. Verify the DomainManager *registerDevice* operation registers the new device indicated by the input registerDevice parameter, when the previously registered device has the same identifier attribute as a previously registered device and the reference to the registered device refers to a nonexistent object. (OE0717)				
2. Locate where the *registerDevice* operation validates the input parameter *registeringDevice* against already registered Devices.	**Pass**: The source code where the *registerDevice* operation validates the input parameter *registeringDevice* against already registered Devices is found. (OE0717) **Fail**: The source code where the *registerDevice* operation validates the input parameter *registeringDevice* against already registered Devices is not found. (OE0717)			

OE_TC_168				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Locate where the *registerDevice* operation verifies the already registered Device is a non-existent object.	**Pass**: The source code where the *registerDevice* operation verifies the already registered Device is a non-existent object is found. (OE0717) **Fail**: The source code where the *registerDevice* operation verifies the already registered Device is a non-existent object is not found. (OE0717)			
4. Verify the *registerDevice* operation will register the device indicated by the registeringDevice parameter.	**Pass:** The *registerDevice* operation registers the device indicated by the registeringDevice parameter. (OE0717) **Fail:** The *registerDevice* operation does not register the device indicated by the registeringDevice parameter. (OE0717)			
C. Verify that the DomainManager *registerDevice* operation writes an ADMINISTRATIVE_EVENT log record when reference to the registered device refers to a nonexistent object. (OE0718)				

OE_TC_168				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Verify that the *registerDevice* operation writes an ADMINISTRATIVE_EVENT log record when the reference to the registered device refers to a nonexistent object.	**Pass:** The *registerDevice* operation writes an ADMINISTRATIVE_EVENT log record when reference to the registered device refers to a nonexistent object. (OE0718) **Fail:** The *registerDevice* operation does not write an ADMINISTRATIVE_EVENT log record when reference to the registered device refers to a nonexistent object. (OE0718)			
End of Test				

Test Recording Log – OE_TC_168			
Step2 (input is validated against registered Devices) Y/N	**Step3** (recognizes that registered Device is non-existent) Y/N	**Step4** (registers the device indicated by the input) Y/N	**Step5** (writes log record about this event) Y/N

Test Summary OE_TC_168

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):

OE0717_____

OE0718 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.38. OE_TC_189 - Application Delegates Implementation of Resource operations

Test Case Number: OE_TC_189

Application

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0127	The application shall delegate the implementation of the inherited Resource operations (runTest, start, stop, configure, and query) to the Application Resource component identified by the application's SAD *assemblycontroller* element (Assembly Controller).
OE0128	The application shall propagate exceptions raised by the application's Assembly Controller's operations.
OE0129	The initialize operation shall not be propagated to the application's components or its Assembly Controller.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-22 through 3-23, Section 3.1.3.2.1.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-14, C-15, C-16, C-24

Test Objective

This test case verifies OE0127, OE0128, and OE0129. The objective of this test is to ensure that the Application interface properly passes requests to the assembly controller for the runTest, start, stop, configure, and query operations. It also verifies that exceptions are properly passed back through the Application interface from the runTest, start, stop, configure, and query operations. This test also ensures that exceptions from the Application interfaces initialize operation are not propagated to the application's components.

Places to Verify

Application interface.

IDL References

Data

```
exception UnknownTest();
exception UnknownProperties (CF::Properties invalidProperties;);
exception StartError { CF::ErrorNumberType errorNumber; string msg;};
exception StopError {CF::ErrorNumberType errorNumber; string msg;};
exception InvalidConfiguration {string msg; CF::Properties invalidProperties;};
```

exception PartialConfiguration {CF::Properties invalidProperties;};
exception InitializeError {CF::StringSequence errorMessages;};

Operations

void runTest (in unsigned long testid, inout CF::Properties testValues)
raises (CF::TestableObject::UnknownTest, CF::UnknownProperties);
void start () raises (CF::Resource::StartError);
void stop () raises (CF::Resource::StopError);
void configure (in CF::Properties configProperties)
raises (CF::PropertySet::InvalidConfiguration, CF::PropertySet::PartialConfiguration);
void query (inout CF::Properties configProperties) raises (CF::UnknownProperties);
void initialize () raises (CF::LifeCycle::InitializeError);

Preconditions

- The Application interface source code files are available.

Test Description

A. Verify that the application delegates the implementation of the inherited Resource operations (runTest, start, stop, configure, and query) to the Application Resource component identified by the application's SAD *assemblycontroller* element (Assembly Controller). (OE0127)
   1. **Untested:** It cannot be determined how the application obtains a reference to the *assemblycontroller* element that Resource operations will be delegated to.
   2. **Untested:** The application's implementation of inherited Resource operations (runTest, start, stop, configure, and query) cannot be located.
   3. **Pass:** The application's implementation of inherited Resource operations (runTest, start, stop, configure, and query) delegates to the Application Resource component identified by the application's SAD *assemblycontroller* element (Assembly Controller).
   4. **Fail:** The application's implementation of inherited Resource operations (runTest, start, stop, configure, and query) does not delegate to the Application Resource component identified by the application's SAD *assemblycontroller* element (Assembly Controller).

B. Verify that the application propagates exceptions raised by the application's Assembly Controller's operations. (OE0128)
   1. **Pass:** The application's implementation of inherited Resource operations (runTest, start, stop, configure, and query) propagates exceptions received from the *assemblycontroller* element (Assembly Controller).
   2. **Fail:** The application's implementation of inherited Resource operations (runTest, start, stop, configure, and query) does not propagate exceptions received from the *assemblycontroller* element (Assembly Controller)

C. Verify that the application's initialize operation is not delegated to the application's components or its Assembly Controller. (OE0129)

1. **Untested:** The application's initialize operation cannot be located.
2. **Pass:** The application's initialize operation is not delegated to the application's components or its Assembly Controller.
3. **Fail:** The application's initialize operation is delegated to the application's components or its Assembly Controller.

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_189				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the application delegates the implementation of the inherited Resource operations (runTest, start, stop, configure, and query) to the Application Resource component identified by the application's SAD *assemblycontroller* element (Assembly Controller). (OE0127)				
1. Determine how the Application interface obtains a reference to the *assemblycontroller* element as specified in the SAD.	**Untested:** It cannot be determined how the Application obtains a reference to the *assemblycontroller* element that Resource operations will be delegated to. (OE0127)		This is used to determine that Resource operations are being properly delegated to the *assemblycontroller* element.	
2. Locate the Application *runTest* operation.	**Untested:** The Application's *runTest* operation cannot be located. (OE0127)			
3. Verify that the *runTest* operation delegates the implementation to the *assemblycontroller* element.	**Pass:** The *runTest* operation delegates to the *assemblycontroller* element. (OE0127) **Fail:** The *runTest* operation does not delegate to the *assemblycontroller* element. (OE0127)		This should be an invocation of the appropriate method on the reference to the *assemblycontroller* element.	
4. Locate the Application *start* operation.	**Untested:** The Application's *start* operation cannot be located. (OE0127)			
5. Verify that the *start* operation delegates the implementation to the *assemblycontroller* element.	**Pass:** The *start* operation delegates to the *assemblycontroller* element. (OE0127) **Fail:** The *start* operation does not delegate to the *assemblycontroller* element. (OE0127)		This should be an invocation of the appropriate method on the reference to the *assemblycontroller* element.	
6. Locate the Application *stop* operation.	**Untested:** The Application's *stop* operation cannot be located. (OE0127)			

OE_TC_189				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Verify that the *stop* operation delegates the implementation to the *assemblycontroller* element.	**Pass:** The *stop* operation delegates to the *assemblycontroller* element. (OE0127) **Fail:** The *stop* operation does not delegate to the *assemblycontroller* element. (OE0127)		This should be an invocation of the appropriate method on the reference to the *assemblycontroller* element.	
8. Locate the Application configure operation.	**Untested:** The Application's *configure* operation cannot be located. (OE0127)			
9. Verify that the *configur*e operation delegates the implementation to the *assemblycontroller* element.	**Pass:** The *configure* operation delegates to the *assemblycontroller* element. (OE0127) **Fail:** The *configure* operation does not delegate to the *assemblycontroller* element. (OE0127)		This should be an invocation of the appropriate method on the reference to the *assemblycontroller* element.	
10. Locate the Application *query* operation.	**Untested:** The Application's *query* operation cannot be located. (OE0127)			
11. Verify that the *query* operation delegates the implementation to the *assemblycontroller* element.	**Pass:** The *query* operation delegates to the *assemblycontroller* element. (OE0127) **Fail:** The *query* operation does not delegate to the *assemblycontroller* element. (OE0127)		This should be an invocation of the appropriate method on the reference to the *assemblycontroller* element.	
B. Verify that the application propagates exceptions raised by the application's Assembly Controller's operations. (OE0128)				

OE_TC_189				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
12. Verify that the *runTest* operation propagates exceptions raised by the application's *assemblycontroller* element.	**Pass:** The application's implementation of *runTest* propagates exceptions received from the *assemblycontroller* element. (OE0128) **Fail:** The application's implementation of *runTest* does not propagate exceptions received from the *assemblycontroller* element. (OE0128)		Same as location determined in step 2. Exceptions thrown by the *assemblycontroller* element should either not be caught or should be re-thrown.	
13. Verify that the *start* operation propagates exceptions raised by the application's *assemblycontroller* element.	**Pass:** The application's implementation of *start* propagates exceptions received from the *assemblycontroller* element. (OE0128) **Fail:** The application's implementation of *start* does not propagate exceptions received from the *assemblycontroller* element. (OE0128)		Same as location determined in step 4. Exceptions thrown by the *assemblycontroller* element should either not be caught or should be re-thrown	
14. Verify that the *stop* operation propagates exceptions raised by the application's *assemblycontroller* element.	**Pass:** The application's implementation of *stop* propagates exceptions received from the *assemblycontroller* element. (OE0128) **Fail:** The application's implementation of *stop* does not propagate exceptions received from the *assemblycontroller* element. (OE0128)		Same as location determined in step 6. Exceptions thrown by the *assemblycontroller* element should either not be caught or should be re-thrown	

OE_TC_189				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
15. Verify that the *configure* operation propagates exceptions raised by the application's *assemblycontroller* element.	**Pass:** The application's implementation of *configure* propagates exceptions received from the *assemblycontroller* element. (OE0128) **Fail:** The application's implementation of *configure* does not propagate exceptions received from the *assemblycontroller* element. (OE0128)		Same as location determined in step 8. Exceptions thrown by the *assemblycontroller* element should either not be caught or should be re-thrown	
16. Verify that the *query* operation propagates exceptions raised by the application's *assemblycontroller* element.	**Pass:** The application's implementation of *query* propagates exceptions received from the *assemblycontroller* element. (OE0128) **Fail:** The application's implementation of *query* does not propagate exceptions received from the *assemblycontroller* element. (OE0128)		Same as location determined in step 10. Exceptions thrown by the *assemblycontroller* element should either not be caught or should be re-thrown	
C. Verify that the application's initialize operation is not delegated to the application's components or its Assembly Controller. (OE0129)				
17. Locate the Application's *initialize* operation.	**Untested:** The application's *initialize* operation cannot be located. (OE0129)			

OE_TC_189				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
18. Verify that the *initialize* operation is not delegated to the application's components or its Assembly Controller.	**Pass:** The application's *initialize* operation is not delegated to the application's components or its Assembly Controller. (OE0129) **Fail:** The application's *initialize* operation is delegated to the application's components or its Assembly Controller. (OE0129)			
End of Test				

Test Recording Log – TE_OE_TC_189		
Step 1-11 (Resource operations delegated)	**Step 11-16** (Assembly Controller exceptions propagated)	**Step 17-18** (App's initialize exceptions not propagated)

Test Summary OE_TC_189

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):

OE0127_____

OE0128 _____

OE0129 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.39. OE_TC_224 - DeviceManager Attributes

Test Case Number: OE_TC_224

DeviceManager attributes

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0468	The readonly label attribute shall contain the device manager's label.
OE0469	The readonly fileSys attribute shall contain the FileSystem associated with this device manager.
OE0470	The readonly deviceConfigurationProfile attribute shall contain a profile element (Profile Descriptor) with a file reference to the device manager's Device Configuration Descriptor (DCD) file.
OE0471	The readonly registeredDevices attribute shall contain a list of devices that have registered with this device manager or a sequence length of zero if no devices have registered with the device manager.
OE0472	The readonly registeredServices attribute shall contain a list of services that have registered with this device manager or a sequence length of zero if no services have registered with the device manager.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-51 to 3-52, Section 3.1.3.2.4.4.2 thru Section 3.1.3.2.4.4.6

Test Objective

This test case verifies OE0468, OE0469, OE0470, OE0471, and OE0472. The objective of this test is to verify a deviceManager interface's label, fileSys, deviceConfigurationProfile, registeredDevices and registeredServices read-only attributes.

Places to Verify

Device Manager interface's source code

IDL References

Data

```
readonly attribute string label;
readonly attribute FileSystem fileSys;
readonly attribute string deviceConfigurationProfile;
readonly attribute DeviceSequence registeredDevices;
```

readonly attribute ServiceSequence registeredServices;

Preconditions

- The DeviceManager interface source code files are available.

Test Description

A. Locate all implementations of the DeviceManager interface. (OE0468, OE0469, OE0470, OE0471, OE0472)
   1. **Fail**: There are no implementations of the DeviceManager interface.
   2. **Pass**: Device source code is found.

For each implementation of the DeviceManager interface found, perform the following:

B. Verify that a readonly "label" attribute exists containing the name of the DeviceManager (OE0468)
   1. **Pass:** The attribute exists and contains the name of the DeviceManager.
   2. **Fail:** The attribute does not exist.
   3. **Fail:** The attribute is not readonly.
   4. **Fail**: The attribute does not contain the name of the DeviceManager.

C. Verify that a readonly "fileSys" attribute exists containing the FileSystem associated with the DeviceManager. (OE0469)
   1. **Pass:** The "fileSys" attribute exists and contains the name of the FileSystem associated with the DeviceManager.
   2. **Fail:** The attribute does not exist.
   3. **Fail:** The attribute is not readonly.
   4. **Fail:** The attribute does not contain the name of the FileSystem associated with the DeviceManager.

D. Verify that a readonly "deviceConfigurationProfile" attribute exists containing a profile descriptor with a file reference to the DCD file associated with the DeviceManager. (OE0470).
   1. **Pass:** The "deviceConfigurationProfile" attribute exists and contains a profile descriptor with a file reference to the DCD associated with the DeviceManager.
   2. **Fail:** The attribute does not exist.
   3. **Fail:** The attribute is not readonly.
   4. **Fail:** The attribute does not contain a file reference to the DCD associated with the DeviceManager.
   5. **Fail**: The DeviceManager's DCD file name is not stored in the "deviceConfigurationProfile" variable.

E. Verify that a readonly "registeredDevices" attribute exists containing a list of the devices that have registered with the DeviceManager. (OE0471)
   1. **Pass:** The attribute exists and contains a list of the devices that have registered with the DeviceManager.
   2. **Fail:** The attribute does not exist.
   3. **Fail:** The attribute is not readonly.

4. **Fail**: The attribute does not contain a list of the devices that have registered with the DeviceManager.
5. **Fail**: The attribute sequence length is not 0 (zero) when no devices have registered.

F. Verify that there exists a readonly "registeredServices" attribute that contains a list of the services that have registered with the DeviceManager. (OE0472)
1. **Pass:** The attribute exists and contains a list of the services that have registered with the DeviceManager.
2. **Fail:** The attribute does not exist.
3. **Fail:** The attribute is not readonly.
4. **Fail**: The attribute does not contain a list of the services that have registered with the DeviceManager.
5. **Fail**: The attribute sequence length is not 0 (zero) when no services have registered.

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_224				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all implementations of the DeviceManager interface. (OE0468, OE0469, OE0470, OE0471, OE0472)				
1. Locate all implementations of the DeviceManager interface in the source code.	**Pass**: DeviceManager source code is found. (OE0468, OE0469, OE0470, OE0471, OE0472) **Fail**: No DeviceManager source code found. (OE0468, OE0469, OE0470, OE0471, OE0472)			
For each implementation of the DeviceManager interface found				
B. Verify that a readonly label attribute exists containing the name of the DeviceManager (OE0468).				
2. Verify that there is a readonly attribute named "label" that is a string variable.	**Pass:** A readonly string variable named "label" exists. (OE0468) **Fail:** A string variable named "label" does not exist. (OE0468) **Fail:** A string variable named "label" is not readonly. (OE0468)		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.	
3. Verify that the attribute named "label" contains the device's label.	**Pass:** A string variable named "label" contains the device's label. (OE0468) **Fail:** A string variable named "label" does not contain the device's label. (OE0468)			
C. Verify that a readonly fileSys attribute exists containing the FileSystem associated with the DeviceManager. (OE0469)				

OE_TC_224				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that there is a readonly attribute named "fileSys" that is a string variable.	**Pass:** A readonly string variable named "fileSys" exists. (OE0469) **Fail:** A string variable named "fileSys" does not exist. (OE0469). **Fail:** A string variable named "fileSys" is not readonly. (OE0469)			
5. Verify that the attribute named "fileSys" contains the path to the FileSystem associated with the DeviceManager.	**Pass:** A string variable named "fileSys" contains the path to the FileSystem associated with the DeviceManager. (OE0469) **Fail:** A string variable named "fileSys" does not contain the path to the FileSystem associated with the DeviceManager. (OE0469)			
D. Verify that a readonly "deviceConfigurationProfile" attribute exists containing a profile descriptor with a file reference to the DCD file associated with the DeviceManager. (OE0470).				
6. Verify that there is a readonly attribute named "deviceConfigurationProfile" that is a string variable.	**Pass:** A readonly string variable named "deviceConfigurationProfile" exists. (OE0470) **Fail:** A string variable named "deviceConfigurationProfile" does not exist. (OE0470) **Fail:** A string variable named "deviceConfigurationProfile" is not readonly. (OE0470)			

OE_TC_224				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Verify that the variable named "deviceConfigurationProfile" contains a profile descriptor with a file reference to the DCD file associated with the DeviceManager.	**Pass:** A string variable named "deviceConfigurationProfile" contains a profile descriptor with a file reference to the DCD file associated with the DeviceManager. (OE0470) **Fail:** A string variable named "deviceConfigurationProfile" does not contain a profile descriptor with a file reference to the DCD file associated with the DeviceManager. (OE0470)			
E. Verify that a readonly registeredDevices attribute exists containing a list of the devices that have registered with the DeviceManager. (OE0471)				
8. Verify that there is a readonly attribute named "registeredDevices" that is a sequence of strings.	**Pass:** A readonly variable named "registeredDevices", consisting of a sequence of strings, exists. (OE0471) **Fail:** A variable named "registeredDevices", consists of a sequence of strings, does not exist. (OE0471) **Fail:** A string variable named "registeredDevices" is not readonly. (OE0471)			

OE_TC_224				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
9. Verify that the variable named "registeredDevices" contains a list of devices that have registered with the DeviceManager.	**Pass:** A variable named "registeredDevices" contains a list of devices that have registered with the DeviceManager. (OE0471) **Fail:** A variable named "registeredDevices" does not contain a list of devices that have registered with the DeviceManager. (OE0471)			
10. Verify that when no devices have registered with the DeviceManager, the sequence named "registeredDevices" is zero length.	**When no devices have registered with the DeviceManager:** **Pass:** A sequence named "registeredDevices" has zero length. (OE0472) **Fail:** A sequence named "registeredDevices" does not have zero length. (OE0472)			
F. Verify that there exists a readonly registeredServices attribute that contains a list of the services that have registered with the DeviceManager. (OE0472)				
11. Verify that there is a readonly attribute named "registeredServices" that is a sequence of strings.	**Pass:** A readonly variable named "registeredServices", consisting of a sequence of strings, exists. (OE0472) **Fail:** A variable named "registeredServices", consists of a sequence of strings, does not exist. (OE0472) **Fail:** A string variable named "registeredServices" is not readonly. (OE0472)			

OE_TC_224				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
12. Verify that the variable named "registeredServices" contains a list of services that have registered with the DeviceManager.	**Pass:** A variable named "registeredServices" contains a list of services that have registered with the DeviceManager. (OE0472) **Fail:** A variable named "registeredServices" does not contain a list of services that have registered with the DeviceManager. (OE0472)			
13. Verify that when no services have registered with the DeviceManager, the sequence named "registeredServices" is zero length.	**When no services have registered with the DeviceManager:** **Pass:** A sequence named "registeredServices" has zero length. (OE0472) **Fail:** A sequence named "registeredServices" does not have zero length. (OE0472)			
End of Test				

Test Recording Log – OE_TC_224						
Step2 (label attribute exists)	**Step3** (label attribute names a device)	**Step4 & 5** (fileSys attribute exists and names a FileSystem)	**Step6 & 7** (deviceConfiguration Profile attribute contains a file ref)	**Step8-10** (registeredDevices has list of registered devices)	**Step11-13** (registeredServices has list of registered services)	**Notes**

Test Summary OE_TC_224

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):

OE0468 _____

OE0469 _____

OE0470 _____

OE0471 _____

OE0472 _____

Failed Items (Section/Step Number):

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.2.40. OE_TC_233 - Application Attributes

Test Case Number: OE_TC_233

Application attributes

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0121	The readonly profile attribute shall contain a profile element (Profile Descriptor) with a file reference to the application's SAD file.
OE0122	This readonly name attribute shall contain the name of the created application.
OE0123	The componentNamingContexts attribute shall contain the list of components' Naming Service Context within the application for those components using CORBA Naming Service.
OE0124	The componentProcessIds attribute shall contain the list of components' process IDs within the Application for components that are executing on a device.
OE0125	The componentDevices attribute shall contain a list of devices, which each component either uses, is loaded on or is executed on.
OE0126	The componentImplementations attribute shall contain the list of components' SPD implementation IDs within the application for those components created.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-22, Section 3.1.3.2.1.4.1 thru Section 3.1.3.2.1.4.6
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-28 thru C-29

Test Objective

This test case verifies OE0121, OE0122, OE0123, OE0124, OE0125, and OE0126. The test objective is to verify the attributes profile, name, componentNamingContexts, componentProcessIds, componentDevices and componentImplementations of an implemented Application. Attributes will be verified for type, content, and availability.

Places to Verify

All implementations of the Application interface.

IDL References

Data

readonly attribute string profile;

readonly attribute string name;
readonly attribute CF::Application::ComponentElementSequence componentNamingContexts;
readonly attribute CF::Application::ComponentProcessIdSequence componentProcessIds;
readonly attribute CF::DeviceAssignmentSequence componentDevices;
readonly attribute CF::Application::ComponentElementSequence componentImplementations;

Preconditions

- The Application interface source code files are available.

Test Description

A. Locate all implementations of the Application interface. (OE0121, OE0122, OE0123, OE0124, OE0125, OE0126)
   1. Fail: There are no implementations of the Application interface.

For each implementation of the Application interface, locate the code that defines the Application class and perform the following:

B. Verify that the readonly profile attribute contains a profile element (Profile Descriptor) with a file reference to the application's SAD file. (OE0121)
   1. **Pass:** The profile attribute is a readonly string that contains a profile element (Profile Descriptor) with a file reference to the application's SAD file.
   2. **Fail:** The profile attribute does not exist.
   3. **Fail:** The profile attribute is not a string.
   4. **Fail:** The profile attribute is not readonly.
   5. **Fail:** The profile attribute does not contain a profile element (Profile Descriptor) with a file reference to the application's SAD file.
C. Verify that the readonly name attribute contains the name of the created application. (OE0122)
   1. **Pass:** The name attribute is a readonly string that contains the name of the created application.
   2. **Fail:** The name attribute does not exist.
   3. **Fail:** The name attribute is not a string.
   4. **Fail:** The name attribute is not readonly.
   5. **Fail:** The name attribute does not contain the name of the created application.
D. Verify that the readonly componentNamingContexts attribute contains the list of components' Naming Service Context within the application for those components using CORBA Naming Service. (OE0123)
   1. **Pass:** The readonly componentNamingContexts attribute contains the list of components' Naming Service Context within the application for those components using CORBA Naming Service.
   2. **Fail:** The componentNamingContexts attribute does not exist.
   3. **Fail:** The componentNamingContexts attribute is not a CF::Application:: ComponentElementSequence type.

4. **Fail:** The componentNamingContexts attribute is not readonly.
5. **Fail:** The componentNamingContexts attribute does not contain a list of components' Naming Service Contexts.

E. Verify that the componentProcessIds attribute contains the list of components' process IDs within the Application for components that are executing on a device. (OE0124)
   1. **Pass:** The componentProcessIds attribute contains the list of components' process IDs within the Application for components that are executing on a device.
   2. **Fail:** The componentProcessIds attribute does not exist.
   3. **Fail:** The componentProcessIds attribute is not a ComponentProcessIdSequence type.
   4. **Fail:** The componentProcessIds attribute is not readonly.
   5. **Fail:** The componentProcessIds attribute does not contain a list of components' process IDs.

F. Verify that the componentDevices attribute contains a list of devices, which each component either uses, is loaded on or is executed on. (OE0125)
   1. **Pass:** The componentDevices attribute contains a list of devices, which each component either uses, is loaded on, or is executed on.
   2. **Fail:** The componentDevices attribute does not exist.
   3. **Fail:** The componentDevices attribute is not a CF::DeviceAssignmentSequence type.
   4. **Fail:** The componentDevices attribute is not readonly.
   5. **Fail:** The componentDevices attribute does not contain a list of devices, which each component either uses, is loaded on, or is executed on.

G. Verify that the componentImplementations attribute contains the list of components' SPD implementation IDs within the application for those components created. (OE0126)
   1. **Pass:** The componentImplementations attribute contains the list of components' SPD implementation IDs within the application for those components created.
   2. **Fail:** The componentImplementations attribute does not exist.
   3. **Fail:** The componentImplementations attribute is not a CF::Application::ComponentElementSequence type.
   4. **Fail:** The componentImplementations attribute is not readonly.
   5. **Fail:** The componentImplementations attribute does not contain the list of components' SPD implementation IDs within the application for those components created.

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_233				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all implementations of the Application interface. (OE0122, OE0123, OE0124, OE0125, OE0126)				
1. Locate and record files that implement the Application interface.	**Fail**: There are no implementations of the Application interface.			
For each implementation of the Application interface, locate the code that defines the Application class and perform the following:				
B. Verify that the readonly profile attribute contains a profile element (Profile Descriptor) with a file reference to the application's SAD file. (OE0121)				
2. Verify that the profile string attribute exists.	**Pass:** The profile attribute does exist. (OE0121) **Fail:** The profile attribute does not exist. (OE0121)			
3. Verify that the profile attribute is a string.	**Pass:** The profile attribute is a string. (OE0121) **Fail:** The profile attribute is not a string. (OE0121)			
4. Verify the profile attribute is readonly.	**Fail:** The profile attribute is readonly. (OE0121) **Fail:** The profile attribute is not readonly. (OE0121)			
5. Verify that the profile attribute contains a profile element (Profile Descriptor) with a file reference to the application's SAD file.	**Pass:** The profile attribute that contains a profile element (Profile Descriptor) with a file reference to the application's SAD file. (OE0121) **Fail:** The profile attribute does not contain a profile element (Profile Descriptor) with a file reference to the application's SAD file. (OE0121)		This value is implementation dependent. It is likely that it will be passed to the constructor or an initialization method of the Application class from the ApplicationFactory	
C. Verify that the readonly name attribute contains the name of the created application. (OE0122)				

OE_TC_233				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Verify that the name string attribute exists.	**Pass:** The name string attribute exists. (OE0122) **Fail:** The name attribute does not exist. (OE0122) **Fail**: The name attribute is not a string. (OE0122)			
7. Verify that the name attribute is readonly.	**Pass:** The name attribute is has no publicly accessible set operation for it. (OE0122) **Fail:** The name attribute has a set operation that is publicly accessible. (OE0122)			
8. Verify that name attribute is set to the name of the application.	**Pass**: The name attribute is set to the name of the application. (OE0122) **Fail**: The name attribute is not set to the name of the application. (OE0122)		This value is implementation dependent. It is likely that it will be passed to the constructor or an initialization method of the Application class from the ApplicationFactory.	
D. Verify that the readonly componentNamingContexts attribute contains the list of components' Naming Service Context within the application for those components using CORBA Naming Service. (OE0123)				
9. Verify that the componentNamingContexts attribute exists and is of type ComponentElementSequence.	**Pass:** The componentNamingContexts attribute exists and is of type ComponentElementSequence. (OE0123) **Fail:** The componentNamingContexts attribute does not exist. (OE0123) **Fail**: The componentNamingContexts attribute is not a CF::Application::ComponentElementSequence type. (OE0123)			
10. Verify that the componentNamingContexts attribute is readonly, i.e. there is no publicly accessible set operation for it.	**Pass:** The componentNamingContexts attribute has no publicly accessible set operation for it. (OE0123) **Fail:** The componentNamingContexts attribute has a publicly accessible operation set operation. (OE0123)			

OE_TC_233				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
11. Verify that componentNamingContexts attribute contains a list of components Naming Service Contexts.	**Pass**: The componentNamingContexts attribute contains a list of components Naming Service Contexts. (OE0123) **Fail**: The componentNamingContexts attribute does not contain a list of components Naming Service Contexts. (OE0123)		This value is implementation dependent. It is likely that it will be passed to the constructor or an initialization method of the Application class from the ApplicationFactory.	
E. Verify that the componentProcessIds attribute contains the list of components' process IDs within the Application for components that are executing on a device. (OE0124)				
12. Verify that the componentProcessIds attribute exists and is of type ComponentProcessIdSequence.	**Pass:** The componentProcessIds attribute exists and is of type ComponentProcessIdSequence. (OE0124) **Fail:** The componentProcessIds attribute does not exist. (OE0124)			
13. Verify that the componentProcessIds attribute is readonly, i.e. there is no publicly accessible set operation for it.	**Pass:** The componentProcessIds attribute there is no publicly accessible set operation for it. (OE0124) **Fail**: The componentProcessIds attribute has a publicly accessible set operation. (OE0124)			
14. Verify that componentProcessIds attribute contains a list of component's process ID's.	**Pass**: The componentNamingContexts attribute contains a list of component's process IDs. (OE0124) **Fail**: The componentNamingContexts attribute does not contain a list of component's process IDs. (OE0124)		This value is implementation dependent. It is likely that it will be passed to the constructor or an initialization method of the Application class from the ApplicationFactory.	
F. Verify that the componentDevices attribute contains a list of devices, which each component either uses, is loaded on or is executed on. (OE0125)				

OE_TC_233				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
15. Verify that the componentDevices attribute exists and is of type DeviceAssignmentSequence.	**Pass:** The componentDevices attribute exists and is of type DeviceAssignmentSequence. (OE0125) **Fail:** The componentDevices attribute does not exist. (OE0125) **Fail**: The componentDevices attribute is not a CF::DeviceAssignmentSequence type. (OE0125)			
16. Verify that the componentDevices attribute is readonly, i.e. there is no publicly accessible set operation for it.	**Pass:** The componentDevices attribute there is no publicly accessible set operation for it. (OE0125) **Fail**: The componentDevices attribute has a publicly accessible set operation for it. (OE0125)			
17. Verify that componentDevices attribute contains a list of devices, which each component either uses, is loaded on, or is executed on.	**Pass**: The componentDevices attribute contains a list of devices, which each component either uses, is loaded on, or is executed on. (OE0125) **Fail**: The componentDevices attribute does not contain a list of devices, which each component either uses, is loaded on, or is executed on. (OE0125)		This value is implementation dependent. It is likely that it will be passed to the constructor or an initialization method of the Application class from the ApplicationFactory.	
G. Verify that the componentImplementations attribute contains the list of components' SPD implementation IDs within the application for those components created. (OE0126)				
18. Verify that the componentImplementations attribute exists and is of type ComponentElementSequence.	**Pass:** The componentImplementations attribute exists and is of type ComponentElementSequence. (OE0126) **Fail:** The componentImplementations attribute does not exist. (OE0126) **Fail**: The componentImplementations attribute is not a CF::Application::ComponentElementSequence type. (OE0126)			

OE_TC_233				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
19. Verify that the componentImplementations attribute is readonly, i.e. there is no publicly accessible set operation for it.	**Pass:** The componentImplementations attribute has no publicly accessible set operation for it. (OE0126) **Fail**: The componentImplementations attribute has publicly accessible a set operation. (OE0126)			
20. Verify that componentImplementations attribute contains a list of components' SPD implementation IDs.	**Pass**: The componentImplementations attribute contains a list of components' SPD implementation IDs. (OE0126) **Fail**: The componentImplementations attribute does not contain a list of components' SPD implementation IDs. (OE0126)		This value is implementation dependent. It is likely that it will be passed to the constructor or an initialization method of the Application class from the ApplicationFactory.	
End of Test				

Test Recording Log – OE_TC_233						
Step1 (Files implementing Application interface)	**Step2-5** (Valid profile attribute)	**Step6-8** (Valid name attribute)	**Step9-11** (Valid componentNamingContexts attribute)	**Step12-14** (Valid componentProcessIds attribute)	**Step15-17** (Valid componentDevices attribute)	**Step18-20** (Valid component Implementations attribute)

Test Summary OE_TC_233

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):

OE0121_____

OE0122_____

OE0123_____

OE0124 _____

OE0125 _____

OE0126_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.41. OE_TC_236 - ApplicationFactory Attributes

Test Case Number: OE_TC_236

ApplicationFactory attributes

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0153	The name attribute shall be identical to the softwareassembly element name attribute of the application's Software Assembly Descriptor file.
OE0154	The readonly softwareProfile attribute shall contain a profile element (Profile Descriptor) with a file reference to the application's SAD file.
OE0155	The readonly identifier attribute shall contain the unique identifier for an ApplicationFactory instance.
OE0156	The identifier shall be identical to the softwareassembly element id attribute of the application factory's Software Assembly Descriptor file.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-27, Section 3.1.3.2.2.4.1 through Section 3.1.3.2.2.4.3
SCA Appendix C: Core Framework IDL	Final / 15 May 2006, Version 2.2.2	Page C-22
SCA Appendix D: Domain Profile	Final / 15 May 2006, Version 2.2.2	

Test Objective

This test case verifies OE0153, OE0154, OE0155 and OE0156. The objective of this test is to verify the ApplicationFactory attributes name, identifier and softwareProfile are correctly defined. The test verifies that the name attribute is identical to the softwareassembly element name attribute of the application's SAD. It verifies that the softwareProfile attribute contains a profile element with a file reference to the application's SAD. It also verifies that the identifier attribute contains the unique identifier for an ApplicationFactory instance and that it be identical to the softwareassembly element id attribute of the application factory's SAD.

Places to Verify

All implementations of the ApplicationFactory interface.

IDL References

Data

readonly attribute string name;
readonly attribute string identifier;

readonly attribute string softwareProfile;

Preconditions

- All source code files implementing the ApplicationFactory interface are available.
- The applications SAD file is available.

Test Description

A. Locate all implementations of the ApplicationFactory interface. (OE0153, OE154, OE155, OE156)
   1. **Untested**: There are no implementations of the ApplicationFactory interface.

For each implementation of the ApplicationFactory interface, locate the code that defines the ApplicationFactory class and perform the following:

B. Verify the readonly name attribute is identical to the softwareassembly element name attribute of the application's SAD file. (OE0153)
   1. **Pass:** The readonly name attribute is identical to the softwareassembly element name attribute of the application's Software Assembly Descriptor file.
   2. **Fail:** The name attribute is not identical to the softwareassembly element name attribute of the application's Software Assembly Descriptor file.
   3. **Fail:** The name attribute is not readonly.

C. Verify the readonly softwareProfile attribute contains a profile element (Profile Descriptor) with a file reference to the application's SAD file. (OE0154)
   1. **Pass:** The readonly softwareProfile attribute contains a profile element with a file reference to the application's SAD file.
   2. **Fail:** The softwareProfile attribute does not contain a profile element with a file reference to the application's SAD file.
   3. **Fail:** The softwareProfile attribute is not readonly.

D. Verify the readonly identifier attribute contains the unique identifier for an ApplicationFactory instance. (OE0155)
   1. **Pass:** The readonly identifier attribute contains the unique identifier for an ApplicationFactory instance.
   2. **Fail:** The identifier attribute does not contain the unique identifier for an ApplicationFactory instance.
   3. **Fail:** The identifier attribute is not readonly.

E. Verify the identifier is identical to the softwareassembly element id attribute of the application factory's SAD file. (OE0156)
   1. **Pass:** The identifier is identical to the softwareassembly element id attribute of the application factory's SAD file.
   2. **Fail:** The identifier is not identical to the softwareassembly element id attribute of the application factory's SAD 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_236				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all implementations of the ApplicationFactory interface. (OE0153, OE0154, OE0155, OE0156)				
1. Perform a keyword search for *ApplicationFactory* on all source code provided by the developer.	The directories are recorded. **Untested:** No results from keyword search. There is no implementation of the *ApplicationFactory* interface. (OE0153, OE0154, OE0155, OE0156)			
2. Examine the source code files returned in Step 1 and search for location where the ApplicationFactory class attributes are defined. Record the file names for the source code.	The implementations of *ApplicationFactory* are identified in the source code, and the file names of the source code are recorded.			
For each *ApplicationFactory* interface found within the OE under test do the following steps				
B. Verify the readonly *name* attribute is identical to the software assembly element name attribute of the application's SAD file. (OE0153)				
3. Search within ApplicationFactory source code and verify that the *name* string attribute exists.	**Pass:** The *name* string attribute exists. (OE0153) **Fail:** The *name* attribute does not exist. (OE0153) **Fail:** the *name* attribute exists but is not a string. (OE0153)			

OE_TC_236				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify the softwareassembly element name attribute of the application's SAD file is maintained in the *name* attribute.	**Pass:** The softwareassembly element name attribute of the application's SAD file is maintained in the *name* attribute. (OE0153) **Fail:** The softwareassembly element name attribute of the application's SAD file is not maintained in the *name* attribute. (OE0153)			
5. Verify the *name* attribute is readonly.	**Pass:** The *name* attribute is has no publicly accessible set operation for it. (OE0153) **Fail:** The *name* attribute has a set operation that is publicly accessible. (OE0153)			
C. Verify the readonly softwareProfile attribute contains a profile element (Profile Descriptor) with a file reference to the application's SAD file. (OE0154)				
6. Search within ApplicationFactory source code and verify that the *softwareProfile* string attribute exists.	**Pass:** The *softwareProfile* string attribute exists. (OE0154) **Fail:** The *softwareProfile* attribute does not exist. (OE0154) **Fail:** The *softwareProfile* attribute exists but is not declared as a string. (OE0154)			
7. Verify that a file reference to the application's SAD file is maintained in the *softwareProfile* attribute.	**Pass:** The *softwareProfile* attribute contains a profile element with a file reference to the application's SAD file. (OE0154) **Fail:** The *softwareProfile* attribute does not contain a profile element with a file reference to the application's SAD file. (OE0154)			

OE_TC_236				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
8. Verify the *softwareProfile* attribute is readonly.	**Pass:** The *softwareProfile* attribute is has no publicly accessible set operation for it. (OE0154) **Fail:** The *softwareProfile* attribute has a set operation that is publicly accessible. (OE0154)			
D. Verify the readonly *identifier* attribute contains the unique identifier for an ApplicationFactory instance. (OE0155)				
9. Search within ApplicationFactory source code and verify that the *identifier* string attribute exists.	**Pass:** The *identifier* string attribute exists. (OE0155 **Fail:** The *identifier* attribute does not exist. (OE0155 **Fail:** The *identifier* attribute exists but is not declared as a string. (OE0155)			
10. Verify that the identifier attribute value only appears once in the softwareassembly element id of the ApplicationFactory's SAD.	**Pass:** The identifier attribute is unique and only appears once. (OE0155) **Fail:** The identifier attribute is not unique and appears more than once. (OE0155)			
11. Verify the *identifier* attribute is readonly.	**Pass:** The *identifier* attribute is has no publicly accessible set operation for it. (OE0155) **Fail:** The *identifier* attribute has a set operation that is publicly accessible. (OE0155)			
E. Verify the identifier is identical to the softwareassembly element id attribute of the application factory's SAD file. (OE0156)				

OE_TC_236				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
12. Verify by code inspection that the identifier attribute is assigned the value of the softwareassembly element id of the ApplicationFactory's SAD file.	**Pass:** The identifier attribute is assigned the value of the softwareassembly element id of the ApplicationFactory's SAD file. (OE0156) **Fail:** The identifier attribute is assigned a value other than the value of the softwareassembly element id of the ApplicationFactory's SAD file. (OE0156)			
End of Test				

Test Recording Log – OE_TC_236								
Step 1 (Directory name location)	**Step 2** (source file name)	**Step 3** (Verify *name* attribute exists)	**Steps 4 & 5** (Verify *name* attribute assigned)	**Step 6** (Verify *softwareProfile* attribute exists)	**Steps 7 & 8** (Verify *softwareProfile* attribute assigned)	**Step 9** (Verify *identifier* attribute exists)	**Steps 10 & 11** (Verify *identifier* attribute assigned)	**Step 9** (Verify *identifier* is unique)

Test Summary OE_TC_236

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):

OE0153_____

OE0154_____

OE0155 _____

OE0156 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.2.42. OE_TC_285 - DeviceManager startup process

Test Case Number: OE_TC_285

DeviceManager startup process

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0481	The device manager shall use the componentinstantiation element's SPD implementation code's stacksize and priority elements, when specified, for the execute operation options parameters.
OE0482	The device manager shall initialize and then configure logical devices that are started by the device manager, after they have successfully registered with the device manager.
OE0483	The device manager shall configure a DCD's componentinstantiation element provided the componentinstantiation element has "configure" readwrite or writeonly properties with values.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-52 to 3-54, Section 3.1.3.2.4.5 Page 3-54, Figure 3-19.
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-31 to C-33.
SCA Appendix D: Dormain Profile	Version 2.2.2 15 May 2006	Pages D-9 and D10, Section D.2.1.6.3 Pages D-36 to D-39, Section D.6.1.3.3

Test Objective

This test case verifies OE0481, OE0482, and OE0483. The objective of this test case is to verify that the Device Manager performs certain parts of its start-up tasks as specified. The tasks to be verified are that the Device Manager

- parses the DCD and SPD files
- uses the *componentinstantiation* element's SPD implementation code's *stacksize* and *priority* elements, when specified, for the *execute* operation options parameters, (corresponds to step 6 of the SCA Figure 3-19)
- initializes logical devices that are started by the Device Manager after they have successfully registered with the DeviceManager, and (corresponds to step 9 of the SCA Figure 3-19)
- configures a DCD's *componentinstantiation* element provided the *componentinstantiation* element has "configure" readwrite or writeonly properties with values. (corresponds to step 10 of the SCA Figure 3-19)

Note: the SCA Figure 3-19, the Device Manager startup scenario, appears on page 3-54 of the SCA Specification v2.2.2.

Places to Verify

Source code of all Device Managers.

IDL References

Operation

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

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);

void initialize ()
raises (CF::LifeCycle::InitializeError);

void configure (in CF::Properties configProperties)
raises (CF::PropertySet::InvalidConfiguration, CF::PropertySet::PartialConfiguration);

XML References

componentinstantiation XML

```
<!ELEMENT componentinstantiation
        ( usagename?
        , componentproperties?
        , findcomponent?
        )>
<!ATTLIST componentinstantiation
        id ID #REQUIRED>
<!ELEMENT usagename (#PCDATA)>
```

```
<!ELEMENT componentproperties
        ( simpleref | simplesequenceref | structref | structsequenceref )+ >
<!ELEMENT findcomponent
        ( componentresourcefactoryref | namingservice )>
<!ELEMENT componentresourcefactoryref
        ( resourcefactoryproperties? )>
<!ATTLIST componentresourcefactoryref
        refid CDATA #REQUIRED>
```

code XML

```
<!ELEMENT code
        ( localfile
        , entrypoint?
        , stacksize?
        , priority?
        )>

<!ATTLIST code
        type CDATA #IMPLIED>
<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
        name CDATA #REQUIRED>
<!ELEMENT entrypoint (#PCDATA)>
<!ELEMENT stacksize (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
```

Preconditions

- All source code files for the DeviceManager must be available. Also all other source code files that are referred to (listed in "include" statement) and/or are inherited by these components must also be available.

Test Description

A. Locate the source code that pertains to the DeviceManager(s) under test. (OE0481, OE0482, OE0483)
   1. **Untested**: The full implementation is not provided.
   2. **Pass**: Startup processing for the DeviceManager is located.
   3. **Fail**: Startup processing for the DeviceManager is not located.

B. Verify that the DeviceManager retrieves each Devices SPD implementation code's stacksize and priority element values, when specified, and passes these values to the Devices as execute operation options (i.e. execparam: ID and format values). (OE0481)
   1. **Pass:** The DeviceManager passes the appropriate execute parameters as expected
   2. **Fail:** The DeviceManager does not pass the appropriate execute parameters as expected

   Note: The values for these SPD elements are optional. However, at a minimum a check should be done to ensure that the core logic necessary to pass these values, (1) should they be provided, (2) exists and (3) is correctly formulated for the DeviceManager under test. Also note, that the XML provided could be in a pre-parsed state. A careful examination to ensure that the pre-parsed format preserves these values (in the event they are provided) must be conducted and that the DeviceManager supplies these values.

C. Verify that only after the device has registered with the DeviceManager, that the DeviceManager initializes and then configures logical devices as specified in its DCD. (OE0482)
   1. **Pass:** The DeviceManager initializes and configures the Devices in the appropriate order when a logical device registers with the DeviceManager.
   2. **Fail:** The DeviceManager does not initialize or configure the Devices in the appropriate order or omits to perform one of these functions when a logical device registers with the DeviceManager.

   Note: The XML provided could be in a pre-parsed state. A careful examination to ensure that the pre-parsed format preserves these values (in the event they are provided) must be conducted and that the DeviceManager uses these values.

D. Verify that the DeviceManager retrieves Devices configurable properties that are readwrite and writeonly as specified in the respective SPD/SCD property files and uses these properties to configure the respective logical device. (OE0483)
   1. **Pass:** The DeviceManager configures each Device specified in its DCD that has "configure" readwrite or writeonly property with value.

2. **Fail:** The DeviceManager does not fetch all of the properties for each of the respective levels that properties could be defined at (SPD Implementation, SPD level, SCD Level) or does not override the value of a property correctly.

Note 1: The XML provided could be in a pre-parsed state. A careful examination to ensure that the pre-parsed format preserves these values (in the event they are provided) must be conducted and that the DeviceManager uses these values.

Note 2: Properties could exist in three levels of the hierarchy. Please refer to Appendix D, section D.6.1.3.3 for the respective levels. The appropriate implementation properties file must also be examined – only one implementation for a given componentinstantiation is launched. Also note, that the rules which govern the union of the properties are also defined in Appendix D, section D.2.1.

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_285				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code that pertains to the DeviceManager(s) under test. (OE0481, OE0482, OE0483)				
1. Locate all implementations of the DeviceManager interface in the source code.	**Untested**: Implementation of the DeviceManager is not found. (OE0481, OE0482, OE0483)			
2. Locate the DeviceManager's startup processing.	**Pass**: Startup processing for the DeviceManager is found. (OE0481, OE0482, OE0483) **Fail**: Startup processing for the DeviceManager is not found. (OE0481, OE0482, OE0483)			
3. Locate within the startup processing of the DeviceManager where XML files are processed for Devices.	**Pass**: XML file processing for Devices is found within the startup processing for the DeviceManager. (OE0481, OE0482, OE0483) **Fail**: XML file processing for Devices is not found within the startup processing for the DeviceManager. (OE0481, OE0482, OE0483)		This may be performed once for all Devices to be executed (that is outside a Device processing loop) or it may be performed for each Device (inside the Device processing loop).	
B. Verify that the DeviceManager retrieves each Devices SPD implementation code's stacksize and priority element values, when specified, and passes this to the Devices as execute operation options (i.e. execparam: ID and format values). (OE0481)				

OE_TC_285				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that this processing includes code for retrieving each Device's SPD implementation code's *stacksize* element value, when specified.	**Pass**: Code indicating the processing of the *stacksize* element and the storing of its value is found. (OE0481) **Fail**: Code indicating the processing of the *stacksize* element and the storing of its value is not found. (OE0481)		The value for the SPD *stacksize* element is optional. However, at a minimum a check should be done to ensure that the core logic necessary to pass this value, (1) should it be provided, (2) exists and (3) is correctly formulated for the DeviceManager under test.	
5. Verify that this processing includes code for retrieving each Device's SPD implementation code's *priority* element value, when specified.	**Pass**: Code indicating the processing of the *priority* element and the storing of its value is found. (OE0481) **Fail**: Code indicating the processing of the *priority* element and the storing of its value is not found. (OE0481)		The value for the SPD *priority* element is optional. However, at a minimum a check should be done to ensure that the core logic necessary to pass this value, (1) should it be provided, (2) exists and (3) is correctly formulated for the DeviceManager under test.	
6. Locate within the startup processing of the DeviceManager where each Device is launched.	**Pass**: The processing where each Device is launched by the DeviceManager is found. (OE0481) **Fail**: The processing where each Device is launched by the DeviceManager is not found. (OE0481)		This action is associated with invocation of the *execute*() operations.	

OE_TC_285				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Verify that this processing uses the retrieved *stacksize* and *priority* values as *execute* operation options	**Pass**: The retrieved values for *stacksize* and *priority* are used as options for launching the Device. (OE0481) **Fail**: The retrieved values for *stacksize* and *priority* are not used as options for launching the Device. (OE0481)			
C. Verify that only after the device has registered with the DeviceManager, that the DeviceManager initializes and then configures logical devices as specified in its DCD. (OE0482)				
8. Locate within the startup processing of the DeviceManager where each Device is initialized and configured.	**Pass**: The processing where each Device is initialized and configured by the DeviceManager is found. (OE0482) **Fail**: The processing where each Device is initialized and configured by the DeviceManager is not found. (OE0482)		These two actions are associated with invocations of the *initialize*() and *configure*() operations.	
9. Verify that this processing occurs after the Device has been launched and it has been registered with the DeviceManager.	**Pass**: The initialize and configure processing occurs after the Device has been launched and registered. (OE0482) **Fail**: The initialize and configure processing does not occur after the Device has been launched and registered. (OE0482)		This action should occur after the *registerDevice*() operation.	

OE_TC_285				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
10. Verify that the DeviceManager initializes and configures logical devices as specified in its DCD.	**Pass**: The DeviceManager initializes and configures logical devices using information from the DCD file. (OE0482) **Fail**: The DeviceManager does not initialize and configure logical devices using information from the DCD file. (OE0482)			
D. Verify that the DeviceManager retrieves Devices configurable properties that are readwrite and writeonly as specified in the respective SPD/SCD property files and uses these properties to configure the respective logical device. (OE0483)				
11. (Based on what was found in step 3) Further locate where the DCD's *componentinstantiation* element is processed.	**Pass**: The processing of the DCD's *componentinstantiation* element is found. (OE0483) **Fail**: The processing of the DCD's *componentinstantiation* element is not found. (OE0483)			
12. Verify that this processing includes code for retrieving the properties which have *readwrite* and *writeonly* mode with their values, if these properties are present.	**Pass**: The processing of the DCD's *componentinstantiation* element includes code for retrieving the properties which have *readwrite* and *writeonly* mode. (OE0483) **Fail**: The processing of the DCD's *componentinstantiation* element does not include code for retrieving the properties which have *readwrite* and *writeonly* mode. (OE0483)			

OE_TC_285				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
13. (Based on what was found in step 8) Verify that the DeviceManager uses these properties in the *configure* operation for the respective logical device.	**Pass**: The DeviceManager uses the *readwrite* and *writeonly* properties in the *configure* operation for the respective logical device. (OE0483) **Fail**: The DeviceManager does not use the *readwrite* and *writeonly* properties in the *configure* operation for the respective logical device. (OE0483)			
End of Test				

Test Recording Log – OE_TC_285				
Steps 1 thru 3 (Locate DeviceManager startup processing)	**Steps 4 & 5** (*stacksize* & *priority* values found)	**Steps 6 & 7** (*stacksize* & *priority* are used to launch Device)	**Steps 8 thru 10** (*initialize* and *configure* come after *execute* and *registerDevice*)	**Steps 11 thru 13** (*readwrite* & *writeonly* are found and used in *configure*)

Test Summary OE_TC_285

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):

OE0481_____

OE0482_____

OE0483_____

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
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
DeviceManager startup process	2	277	B.2.42.	OE_TC_285

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
DeviceManager's execparam properties	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
DomainManager :: unregisterService client-side	2	38	B.2.6.	OE_TC_052
DomainManager :: unregisterService raises UnregisterError	2	55	B.2.9.	OE_TC_055

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
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
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

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
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
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

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
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
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

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
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 testValues parameter	1	201	B.1.25.	OE_TC_045
End of Table				