

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

## **Software Communications Architecture v2.2.2 Manual Application Software Test Description v2.3A**

### **Appendix B: Test Procedures**

**13 April 2022**



**Prepared for:**

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

**Prepared by:**

Joint Tactical Networking Center Test and Evaluation Laboratory

## Table of Contents

<b>APPENDIX B TEST CASES .....</b>	<b>B-4</b>
<b>B.1 APP_TC_001 – LogProducer.....</b>	<b>B-6</b>
<b>B.2 APP_TC_002 – LifeCycle::initialize raise InitializeError .....</b>	<b>B-14</b>
<b>B.3 APP_TC_003 – PortSupplier::GetPort() raises UnknownPort.....</b>	<b>B-20</b>
<b>B.4 APP_TC_004 – SCD describes Uses and Provides Ports .....</b>	<b>B-25</b>
<b>B.5 APP_TC_005 – SAD defines “external” interfaces .....</b>	<b>B-33</b>
<b>B.6 APP_TC_006 – ResourceFactory::releaseResource().....</b>	<b>B-39</b>
<b>B.7 APP_TC_007 – ResourceFactory::shutdown raises ShutdownFailure .....</b>	<b>B-45</b>
<b>B.8 APP_TC_008 – AEP Mandatory OS Services.....</b>	<b>B-52</b>
<b>B.9 APP_TC_009 – TestableObject::runTest() Properties.....</b>	<b>B-63</b>
<b>B.10 APP_TC_010 – TestableObject::runTest().....</b>	<b>B-70</b>
<b>B.11 APP_TC_011 – PropertySet::configure().....</b>	<b>B-76</b>
<b>B.12 APP_TC_012 – Port::disconnectPort raises InvalidPort .....</b>	<b>B-84</b>
<b>B.13 APP_TC_013 – PortSupplier::getPort().....</b>	<b>B-90</b>
<b>B.14 APP_TC_014 – Port::connectPort raises InvalidPort.....</b>	<b>B-95</b>
<b>B.15 APP_TC_015 – Port::connectPort raises OccupiedPort.....</b>	<b>B-101</b>
<b>B.16 APP_TC_016 – TestableObject::runTest raises Exceptions .....</b>	<b>B-107</b>
<b>B.17 APP_TC_017 – Resource::Identifier.....</b>	<b>B-116</b>
<b>B.18 APP_TC_018 – Domain Profile.....</b>	<b>B-122</b>
<b>B.19 APP_TC_019 – LifeCycle::releaseObject().....</b>	<b>B-130</b>
<b>B.20 APP_TC_020 – AEP, Ada Functionality .....</b>	<b>B-136</b>
<b>B.21 APP_TC_021 – High Order Languages .....</b>	<b>B-141</b>
<b>B.22 APP_TC_022 – AEP, Abnormal Termination .....</b>	<b>B-146</b>
<b>B.23 APP_TC_024 – Minimum CORBA.....</b>	<b>B-152</b>
<b>B.24 APP_TC_025 – PropertySet::query().....</b>	<b>B-164</b>
<b>B.25 APP_TC_026 – Base Application Interfaces .....</b>	<b>B-173</b>
<b>B.26 APP_TC_027 – ResourceFactory::createResource.....</b>	<b>B-183</b>
<b>B.27 APP_TC_028 – Core Framework Interfaces .....</b>	<b>B-190</b>
<b>B.28 APP_TC_029 – Producer and Consumer of Events.....</b>	<b>B-197</b>
<b>B.29 APP_TC_030 – AEP, POSIX::kill() .....</b>	<b>B-204</b>
<b>B.30 APP_TC_031 – ResourceFactory::Identifier .....</b>	<b>B-208</b>

---

<b>B.31</b>	<b>APP_TC_032 – Resource::start() raises StartError .....</b>	<b>B-214</b>
<b>B.32</b>	<b>APP_TC_033 – Resource::stop raises StopError.....</b>	<b>B-219</b>
<b>B.33</b>	<b>APP_TC_034 – Resource::stop() not permanent .....</b>	<b>B-225</b>
<b>B.34</b>	<b>APP_TC_035 – ResourceFactory::createResource raises CreateResourceFailure .....</b>	<b>B-231</b>
<b>B.35</b>	<b>APP_TC_036 – Domain Profile files .....</b>	<b>B-237</b>
<b>B.36</b>	<b>APP_TC_037 – Standard arguments for executable components.....</b>	<b>B-265</b>
<b>B.37</b>	<b>APP_TC_039 – Requirements for IDL mappings .....</b>	<b>B-270</b>
<b>B.38</b>	<b>APP_TC_040 – General Rules (Legacy Software) .....</b>	<b>B-281</b>
<b>B.39</b>	<b>APP_TC_041 – OS Services (File access).....</b>	<b>B-287</b>
<b>B.40</b>	<b>APP_TC_042 – OS Services (SIGQUIT).....</b>	<b>B-292</b>
<b>B.41</b>	<b>APP_TC_043 – Registering with the Naming Service.....</b>	<b>B-298</b>
<b>B.42</b>	<b>APP_TC_044 – Usage of stringified IORs.....</b>	<b>B-303</b>

## APPENDIX B TEST CASES

This Appendix contains the test case procedures including the test case name, test objective, preconditions, parameters, test description, the name of the related automated test name (if relevant), and the test requirement(s) being tested.

The Manual Test Steps table contains the test description as the section ‘headers’ followed by the more detailed manual test steps. The test description provides ‘what’ is being tested – not the ‘how’. Each table row contains the steps of what the tester needs to do (such as examine, verify, locate, etc.). The five columns of the table seen in the manual test steps are Steps, Expected Results, Actual Results, Comments and Test Results.

1. **Column 1 (Steps)** – Contains what the tester is to do to verify the requirement(s).
2. **Column 2 (Expected Results)** – Contains the expected consequence that results from the action of the first column.
3. **Column 3 (Actual Results)** – Provides the location where the tester can record the results when Column 1 is performed. When the tester needs additional space to record test results, they can use the Test Recording Log, explained below.
4. **Column 4 (Comments)** – Contains any additional information to aid the tester in the running of the test.
5. **Column 5 (Test Result)** - Provides the location where the tester records the results of the manual steps. The results would be Pass, Fail, Untested or N/A.

The definitions for the various test results are as follows:

- **N/A:** The requirement is not applicable to this component.
- **Untested:** The test could not be completed because required information is missing. If the test is a final test and the developer cannot provide the missing information this becomes a failure.
- **Fail:** The expected results of a step were not met.
- **Pass:** The expected results were met. This step passes. If it is the last step for a requirement and all the steps for that requirement passed then the requirement passes.

To aid in the execution of the test and provide test artifacts, each test case has a Test Recording Log used to record test information. In addition to being used as a test artifact, the Test Recording Log is useful for providing verification of test results. The Test Recording Log is also very helpful when there are many items that must be verified by the tester. For example, if a manual step requires the tester to locate several properties files (PRF) then, the tester can write down the names of all the files used in the test.

At the end of each test case, there is a summary page listing each requirement that was verified through the course of the test case. The tester is encouraged to provide the results of each requirement, which will serve as a quick summary of the status for each requirement. This is also where the participants of the test will sign their names and indicate the date that the test was run.

To perform a test of an application, record the names of the operating system (OS), the operating environment (OE), the CORBA Object Request Broker (ORB), the Core Framework (CF) and, the application.

OS	_____
OE	_____
ORB	_____
CF	_____
Application	_____

## B.1 APP\_TC\_001 – LogProducer

### Test Case Number: APP\_TC\_001

Log Producer

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0011	A log producer shall only output log records that contain an enabled <code>CosLwLog::LogLevel</code> value.
AP0012	Log producers shall use their component identifier attribute in the <code>producerId</code> field of the <code>CosLwLog::ProducerLogRecord</code> .
AP0013	Log producers and CF components that are required by this specification to write log records shall operate normally in the absence of a log service or in the case where the connections to a log are nil or an invalid reference.
AP0747	Log producers shall implement a configure property which is a CF Properties type with an id of “PRODUCER_LOG_LEVEL” and a value that is a <code>CosLwLog::LogLevelSequence</code> .

### References

Document Name	Version/Date	Location (Pages,Section)
SCA	Version 2.2.2; 05-15-2006	Page 3-2

### Test Objective

This test case verifies AP0011, AP0012, AP0013 and AP0747. The objective of this test is to verify that the application properly outputs enabled `LogLevel` log records and does not output disabled `LogLevel` log records. In addition it is the objective of this test that configure properties are implemented with an id of “PRODUCER\_LOG\_LEVEL” and a value that is a `CosLwLog::LogLevelSequence`.

### Places to Verify

Verify in components that write entries to the log.

### IDL References

#### Data

```
struct ProducerLogRecord {  
    string producerId;  
    string producerName;  
    LogLevel level;
```

```
    string logData; }  
// The constants  
// represent the valid values for LogLevel  
// The list of constants may be expanded  
const unsigned short SECURITY_ALARM = 1;  
const unsigned short FAILURE_ALARM = 2;  
const unsigned short DEGRADED_ALARM = 3;  
const unsigned short EXCEPTION_ERROR = 4;  
const unsigned short FLOW_CONTROL_ERROR = 5;  
const unsigned short RANGE_ERROR = 6;  
const unsigned short USAGE_ERROR = 7;  
const unsigned short ADMINISTRATIVE_EVENT = 8;  
const unsigned short STATISTIC_REPORT = 9;  
// Values ranging from 10 to 26 are reserved for  
// program debugging.
```

### Operations

```
oneway void write_records(in ProducerLogRecordSequence records);  
oneway void write_record(in ProducerLogRecord record);
```

### Preconditions

- Log source code files are available.

### Test Description

- A. Identify all the log producer components(*write\_record* and *write\_records* operations) implemented in the application. (AP0011, AP0012, AP0013, AP0747)
  - a. **Untested:** Unable to locate log producer components.
- B. For each log producer component(*write\_record* and *write\_records* operations) identified in Step A, verify the following:
  1. Verify the component identifier attribute is stored in the producerId field of the Log Record (AP0012).
    - a. **Pass:** The component identifier is stored in the producerId field.
    - b. **Fail:** This component identifier is not stored in the producerId field.
  2. Verify that the component works properly when it is not connected to a log service (AP0013).
    - a. **Pass:** The component will continue to work properly in the absence of a log service.

- b. **Pass:** The component handles any exception resulting from the lack of a log service and continues to operate normally.
  - c. **Fail:** The component will not continue to work properly in the absence of a log service
- 3. Verify that only enabled LogLevel values are output (AP0011).
  - a. **Pass:** Only enabled log levels are written.
  - b. **Fail:** The component does not verify the LogLevel value before writing a log record.
  - c. **Fail:** Log messages with a disabled LogLevel value are output.
- C. Verify that a *configure* property with an id of “PRODUCER\_LOG\_LEVEL” using the PropertySet interface is implemented (AP0747).
  - 1. **Pass:** The *configure* property with an id of “PRODUCER\_LOG\_LEVEL” using the PropertySet interface is implemented.
  - 2. **Fail:** The *configure* property with an id of “PRODUCER\_LOG\_LEVEL” using the PropertySet interface is not implemented.



## Manual Test Steps

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

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

APP_TC_001				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify all the log producer components (<i>write_record</i> and <i>write_records</i> operations) implemented in the application. (AP0011, AP0012, AP0013, AP0747)</b>				
1. Locate and record files that invoke the <i>write_record</i> or <i>write_records</i> operations.	<b>Pass:</b> Log producing components are found. (OE0011, OE0012, OE0013, OE0747)  <b>Untested:</b> No <i>write_record</i> or <i>write_records</i> operations are found. (OE0011, OE0012, OE0013, OE0747)			
2. Confirm that the found operations are from CosLwLog.	<b>Pass:</b> The found operations are from the CosLwLog. (AP0011, AP0012, AP0013, AP0747)  <b>Fail:</b> The found operations are not from the CosLwLog. (AP0011, AP0012, AP0013, AP0747)			
<b>B. For each log producer component (<i>write_record</i> and <i>write_records</i> operations) identified in Step A, verify the following:</b>				
<b>1. Verify the component identifier attribute is stored in the producerId field of the Log Record (AP0012).</b>				
3. Find the code that sets the producerId field in the CosLwLog::ProducerLogRecord, and verify that it is set to its component identifier.	<b>Pass:</b> The component identifier is stored in the producerId field. (AP0012)  <b>Fail:</b> This component identifier is not stored in the producerId field. (AP0012)		This could be done as a preset or dynamically, depending on how log records are produced.	
<b>2. Verify that the component works properly when it is not connected to a log service (AP0013).</b>				

APP_TC_001				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Verify that the component does not fail if there is no log service.	<p><b>Pass:</b> The component does not fail in the absence of a log service. (AP0013)</p> <p><b>Pass:</b> The component handles any exception resulting from the lack of a log service and continues to operate normally.. (AP0013)</p> <p><b>Fail:</b> The component fails in the absence of a log service. (AP0013)</p>		<p>The software developer is the best source for this information</p> <p>Some possible code checks could involve: Catching an exception resulting from no log service or verifying that the log service object exists before trying to use it are two ways.</p>	
<b>3. Verify that only enabled LogLevel values are output (AP0011).</b>				
5. Verify that the log record(s) passed to <i>write_record</i> and <i>write_records</i> are of type <i>CosLwLog::ProducerLogRecord</i> .	<p><b>Pass:</b> The log record(s) are of type <i>CosLwLog::ProducerLogRecord</i>. (AP0011)</p> <p><b>Fail:</b> The log record(s) are not of type <i>CosLwLog::ProducerLogRecord</i>. (AP0011)</p>		<p><i>CosLwLog::ProducerLogRecord</i> is a struct that looks like this</p> <pre>struct ProducerLogRecord{     string producerId;     string producerName;     LogLevel level;     string logData; };</pre>	
6. Verify that the <i>LogLevel</i> is checked and that <i>write_record()</i> or <i>write_records()</i> are called only for enabled log levels.	<p><b>Pass:</b> Log records are only written for the log levels that are enabled. (AP0011)</p> <p><b>Fail:</b> The component does not verify the <i>LogLevel</i> value before writing a log record. (AP0011)</p>			
<b>C. Verify that a configure property with an id of “PRODUCER_LOG_LEVEL” using the PropertySet interface is implemented (AP0747).</b>				

APP_TC_001				
Steps	Expected Results	Actual Results	Comments	Test Result
7. Verify that there exists a configure property with an id of “PRODUCER_LOG_LEVEL” and a value that is a CosLwLog::LogLevelSequence.	<p><b>Pass:</b> The proper configure property is implemented. (AP0747)</p> <p>(The property exists and input values are validated before it is set by the configure operation. The values passed in to configure are: CosLwLog::LogLevelSequence)</p> <p><b>Fail:</b> The configure property is not implemented. (AP0747)</p>			
End of Test				

Test Recording Log – APP_TC_001					
Step 1 (Files writing Logs)					
Step 2 (Found files are from CosLwLog) (Y/N)	Step 2 (Log producing Component) (Y/N)	Step 3 (No log service) (Y/N)	Step 4 (log record type) (Y/N)	Step 5 (LogLevel) (Y/N)	Step 6 (Configure operation) (Y/N)

Test Recording Log – APP_TC_001					
Step 1 (Files writing Logs)					
Step 2 (Found files are from CosLwLog) (Y/N)	Step 2 (Log producing Component) (Y/N)	Step 3 (No log service) (Y/N)	Step 4 (log record type) (Y/N)	Step 5 (LogLevel) (Y/N)	Step 6 (Configure operation) (Y/N)

## Test Summary APP\_TC\_001

Once testing is complete for every component of the application 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):

AP0011 \_\_\_\_\_

AP0012 \_\_\_\_\_

AP0013 \_\_\_\_\_

AP0747 \_\_\_\_\_

### Failed Items (Section/StepNumber):

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

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

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

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

## B.2 APP\_TC\_002 – LifeCycle::initialize raise InitializeError

**Test Case Number:** APP\_TC\_002

LifeCycle::initialize raises InitializeError

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0074	The initialize operation shall raise an InitializeError exception when an initialization error occurs.

### References

Document Name	Version/Date	Location (Pages,Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-11 – 3-12

### Test Objective

This test case verifies AP0074. The objective of this test is to verify the compliance of the InitializeError exception (AP0074) which is raised by the LifeCycle::initialize operation. The InitializeError exception is raised by the initialize operation when an initialization error occurs during component initialization. This test verifies the InitializeError exception is implemented in the resource(s) component of an application.

### Places to Verify

Verify in resources that use the LifeCycle interface to instantiate, startup, tear down and/or terminate components.

### IDL References

#### Operations

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

#### Exceptions

```
exception InitializeError {  
    CF::StringSequence errorMessages; };
```

**Preconditions**

- The application source code files are available.

**Test Description**

- A. Identify all resources implemented in the application.
- B. For each resource component identified in Step A, identify the resource(s) that call(s) *LifeCycle::initialize* operation.
  1. **Fail:** There are no instances of the *LifeCycle::initialize* call. Stop test.
- C. For each *LifeCycle::initialize* operation implemented, verify that the operation handles initialize error by throwing an *InitializeError* exception during component initialization. (AP0074)
  1. **Pass:** For each *LifeCycle::initialize* operation called, there is an error handler to throw the *InitializeError* exception.
  2. **Fail:** *LifeCycle::initialize* operation(s) does not throw the *InitializeError* exception when initialization error occurs.

## 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.

APP_TC_002				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify all resources implemented in the application.</b>				
1. Locate, open the SAD file for the application and write the SAD file name in the Test Recording Log sheet at the end of the test.	SAD file opens and the file name is written in the log.			
2. Identify the name of all resource components declared in the SAD file. Record the component name.	Resources are declared in the <componentfile> element of the SAD file.		Resource components are identified in the following element of the SAD file. Do not include device SPDs  <componentfiles> <componentfile> <localfile name=*.spd.xml>	
<b>B. For each resource component identified in Step A, identify the resource(s) that call(s) LifeCycle::initialize operation.</b>				
3. Perform a keyword search for the source file where the LifeCycle::initialize operation is implemented in each resource found in Section A.	Search function returns the name(s) of the source files containing LifeCycle::initialize operation.		For C++ development, the source files are *.cpp files.	
4. Record the source file(s).	<b>Fail:</b> No LifeCycle::initialize operation calls returned.			
<b>C. For each LifeCycle::initialize operation implemented, verify that the operation handles initialize error by throwing an <i>InitializeError</i> exception during component initialization. (AP0074)</b>				



APP_TC_002				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Locate the <i>LifeCycle::initialize</i> operation block and inspect the code to determine that there is an error handler to raise an <i>InitializeError</i> exception for errors during component initialization.	<b>Pass:</b> For each <i>LifeCycle::initialize</i> operation called, there is an error handler to throw the <i>InitializeError</i> exception. (AP0074)  <b>Fail:</b> One or more <i>LifeCycle::initialize</i> operation(s) called does not throw the <i>InitializeError</i> exception. (AP0074)			
End of Test				

Test Recording Log – APP_TC_002			
SAD file:			
Step 2. Resource Component	Step 4. Source file	Step 5. InitializeError exception exists (Yes or No)	Notes

## Test Summary APP\_TC\_002

Once testing is complete for every component of the application 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):**

AP0074 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

### B.3 APP\_TC\_003 – PortSupplier::GetPort() raises UnknownPort

#### Test Case Number: APP\_TC\_003

PortSupplier::getPort raises UnknownPort

#### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0090	The <i>getPort</i> operation shall raise an <i>UnknownPort</i> exception if the port name is invalid.

#### References

Document Name	Version/Date	Location (Pages, Section)
SCA	2.2.2 05-15-2006	Page 3-26, Section 3.1.3.1.4.5.1.5

#### Test Objective

This test case verifies AP0090. The objective of this test is to verify the compliance of the *UnknownPort* exception which is raised by the *getPort* operation when it encounters an invalid port name.

#### Places to Verify

Verify in any Resource or component which automatically inherits the PortSupplier interface.

#### IDL References

##### Operation

Object getPort ( in string name )  
raises (CF::PortSupplier::UnknownPort);

##### Exception

exception UnknownPort { };

#### Preconditions

- The *getPort()* operation source code files are available.

## Test Description

- A. Find all implementations of the *getPort* operation (AP0090).
  - 1. **Fail:** Unable to locate any *getPort* operation code.
- B. For every *getPort* implementation, verify that the *getPort* operation raises an *UnknownPort* exception if the port name is invalid (AP0090).
  - 1. **Pass:** The *getPort* operation raises an *UnknownPort* exception when an invalid port name is encountered.
  - 2. **Fail:** The *getPort* operation does not raise an *UnknownPort* exception when an invalid port name is encountered.

## Manual Test Steps

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

2. The Test Recording Log is intended to record long comments and lists of SPD files, SCD files, PRF, or other file/data.

APP_TC_003				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Find all implementations of the <i>getPort</i> operation (AP0090).</b>				
1. Perform a keyword search for <i>getPort</i> operation. Record the filename of those files with the implementations.	<b>Fail:</b> No PortSupplier::getPort implementations found.		The application software engineer can be of assistance in locating the source code for the PortSupplier. The best clue available in the absence of the engineer is the name of the executable.	
<b>B. For every <i>getPort</i> implementation, verify that the <i>getPort</i> operation raises an <i>UnknownPort</i> exception if the port name is invalid (AP0090).</b>				
2. Examine the <i>getPort</i> operation to find where the input port name is verified.	<b>Fail:</b> The <i>getPort</i> operation does not verify port names. (AP0090)  <b>Pass:</b> The <i>getPort</i> operation verifies that the port name is valid. (AP0090)		The application software engineer can be of assistance in locating the source code for where the port name is verified.	
3. Verify that the <i>getPort</i> operation raises an <i>UnknownPort</i> exception if the port name is invalid.	<b>Fail:</b> The <i>getPort</i> operation does not raise an <i>UnknownPort</i> exception when the port name is invalid. (AP0090)  <b>Pass:</b> The <i>getPort</i> operation raises an <i>UnknownPort</i> exception when the port name is invalid. (AP0090)			
<b>End of Test</b>				

Test Recording Log – APP_TC_003		
Step 1 (source code files for <i>getPort</i> )	Step 2 Input port name is verified? (Y)es or (N)o	Step 3 (Source Code files of failed <i>getPort</i> operation tests)

## Test Summary APP\_TC\_003

Once testing is complete for every component of the application 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):**

AP0090 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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



## B.4 APP\_TC\_004 – SCD describes Uses and Provides Ports

### Test Case Number: APP\_TC\_004

Interface descriptor

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0614	Interfaces provided by a component shall be described in a Software Component Descriptor file as provides ports.
AP0615	Interfaces required by a component shall be described in a Software Component Descriptor file as uses ports.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 2-6, Figure 2-3; Page 3-6 thru 3-8, Section 3.1.3.1.1; Page 3-12, Section 3.1.3.1.4, Page 3-96, Section 3.2.2
SCA, AppendixD: Domain Profile	Version 2.2.2 05-15-2006	Pages D-4 thru D-13, Section D.3; Pages D-28 thru D-47, Sections D.5 and D.6

### Test Objective

This test case verifies AP0614 and AP0615. The objective of this test is to verify that the interfaces which are provided or required for a deployed component within the system are correctly described in a Software Component Descriptor (SCD) as uses and provides port definitions in the interfaces element.

### Places to Verify

The SCD files of the Domain Profile and the Core Framework documentation.

### IDL References

None.

## Preconditions

- Though not required, test case APP\_TC\_036 can provide a listing (or diagram) of all *provides* and *uses* ports by component for all components is available from the test results. The *provides* and *uses* ports information are only available from the SCD files, which is where the test case process gets the information.
- The developer's Interface Control Document (ICD) or other interface documentation of the Core Framework (CF) is available.

## Test Description

A. Identify each component that provides interfaces based on the developer's interface documentation. (AP0614)

1. **Untested:** No interface documentation is provided by the developer.
2. **N/A:** No components are found that provide interfaces.

For each component in the interface documentation that provides interfaces perform step B:

B. Verify that the identified interfaces are *provides* ports as described in the component's SCD (from APP\_TC\_036 results). (AP0614)

1. **Pass:** The *provides* ports of the SCD match those described as provided interfaces in the interface documentation.
2. **Fail:** The *provides* ports of the SCD does not match those described as provided interfaces in the interface documentation.

C. Identify each component that requires interfaces based on the developer's interface documentation. (AP0615)

1. **Untested:** No interface documentation is provided by the developer.
2. **N/A:** No components are found that require interfaces.

For each component in the interface documentation that requires interfaces perform step D:

D. Verify that the identified interfaces are *uses* ports as described in the component's SCD (from APP\_TC\_036 results). (AP0615)

1. **Pass:** The *uses* ports of the SCD match those described as required interfaces in the interface documentation.
2. **Fail:** The *uses* ports of the SCD does not match those described as required interfaces in the interface documentation.

## 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. The information from APP\_TC\_036 shows *provides* and *uses* ports of the AP.

APP_TC_004				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify each component that provides interfaces based on the developer's interface documentation. (AP0614)</b>				
1. Identify each component which provides interfaces based on the developer's interface documentation.	<b>Untested:</b> No interface documentation is provided by the developer. (AP0614)  <b>N/A:</b> No components are found which provide interfaces. There is no way to verify entries found in the SCD. (AP0614)			
2. Identify and list the provided interfaces for the components as described by the interface documentation.	A list of components and their provided interfaces.			
3. Locate and list the source code for the components that provide interfaces as described by the interface documentation. Use the list from <b>Step 2</b> as needed.	A list of components' source code that provide interfaces.			
<b>For each component in the interface documentation which provides interfaces, perform steps 4 through 6 :</b>				
<b>B. Verify that the identified interfaces are <i>provides</i> ports as described in the component's SCD (from APP_TC_036 results). (AP0614)</b>				

APP_TC_004				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Find the corresponding component in the list (or diagram) of <i>provides</i> ports generated from the APP_TC_036 test case.	<p><b>Pass:</b> A corresponding component is found in the list (or diagram). (AP0614)</p> <p><b>Fail:</b> A corresponding component is not found in the list (or diagram). (AP0614)</p>		<p>A corresponding component would be one with the same name or a very similar name.</p> <p>Should an undocumented component be found in the APP_TC_036 results, please report it to the developer and make a note that the interface documentation is incomplete and why.</p>	
5. Identify and list the <i>provides</i> ports for the corresponding component found in <b>Step 4</b> . This should come from the list (or diagram) generated from the APP_TC_036 test case.	<p><b>Pass:</b> A list of <i>provides</i> port(s) is found. (AP0614)</p> <p><b>Fail:</b> No list of <i>provides</i> port(s) is found. (AP0614)</p>		<p>Should an undocumented provides port be found in the APP_TC_036 results, please report it to the developer and make a note that the interface documentation is incomplete and why.</p>	
6. Match the list of provides interfaces created in <b>Step 3</b> with the list of <i>provides</i> ports created in <b>Step 5</b> .	<p><b>Pass:</b> The list of provided interfaces and <i>provides</i> ports match. (AP0614)</p> <p><b>Fail:</b> The list of provided interfaces and <i>provides</i> ports do not match. (AP0614)</p>		<p>Matching the source code provided interfaces with SCD <i>provides</i> ports.</p>	
<b>C. Identify each component that requires interfaces based on the developer's interface documentation. (AP0615)</b>				
7. Identify each component which requires interfaces based on the developer's interface documentation.	<p><b>Untested:</b> No interface documentation is provided by the developer. (AP0615)</p> <p><b>N/A:</b> No components are found which require interfaces. There is no way to verify entries found in the SCD. (AP0615)</p>			

APP_TC_004				
Steps	Expected Results	Actual Results	Comments	Test Result
8. Identify and list the required interfaces for the component as described by the interface documentation.	A list of interface documentation described required interfaces.			
9. Locate and list the source code for the components that require interfaces as described by the interface documentation.	A list of components' source code that use interfaces.			
<b>For each component in the interface documentation which require interfaces, perform steps 10 through 12 :</b>				
<b>D. Verify that the identified interfaces are <i>uses</i> ports as described in the component's SCD (from APP_TC_036 results). (AP0615)</b>				
10. Find the corresponding component in the list (or diagram) of <i>uses</i> ports generated from the APP_TC_036 test case.	<p><b>Pass:</b> A corresponding component is found in the list (or diagram). (AP0615)</p> <p><b>Fail:</b> A corresponding component is not found in the list (or diagram). (AP0615)</p>		<p>A corresponding component would be one with the same name or a very similar name.</p> <p>Should an undocumented component be found in the APP_TC_036 results, please report it to the developer and make a note that the interface documentation is incomplete and why.</p>	
11. Identify and list the <i>uses</i> ports for the corresponding component found in <b>Step 10</b> . This should come from the list (or diagram) generated from the APP_TC_036 test case.	<p><b>Pass:</b> A list of <i>uses</i> port(s) is found. (AP0615)</p> <p><b>Fail:</b> No list of <i>uses</i> port(s) is found. (AP0615)</p>		<p>Should an undocumented required port be found in the APP_TC_036 results, please report it to the developer and make a note that the interface documentation is incomplete and why.</p>	

APP_TC_004				
Steps	Expected Results	Actual Results	Comments	Test Result
12. Match the list created in <b>Step 9</b> with the list created in <b>Step 11</b> .	<b>Pass:</b> The list of required interfaces and <i>uses</i> ports match. (AP0615)  <b>Fail:</b> The list of required interfaces and <i>uses</i> ports do not match. (AP0615)		Matching the documented required interfaces with SCD <i>uses</i> ports.	
End of Test				

<b>Test Recording Log – APP_TC_004</b>			
<b>Step 2</b> (documented provided interfaces)	<b>Step 3</b> (Component source code provided interface)	<b>Step 5</b> ( <i>provides</i> ports from the SCD files)	<b>Step 6</b> (Component source code provided interfaces & SCD <i>provides</i> ports match – Y/N?)
<b>Step 8</b> (documented required interfaces)	<b>Step 9</b> (Component source code required interface)	<b>Step 11</b> ( <i>uses</i> ports from the SCD files)	<b>Step 12</b> (Component source code required interfaces & SCD <i>uses</i> ports match – Y/N?)

## Test Summary APP\_TC\_004

Once testing is complete for every component of the application 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):**

AP0614 \_\_\_\_\_

AP0615 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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



## B.5 APP\_TC\_005 – SAD defines “external” interfaces

**Test Case Number:** APP\_TC\_005

Application – General Rules

### Requirements

SCA Tag	Requirement Text
AP0616	An application interface shall be referenced in the application’s SAD <i>externalports</i> element, and thus declared "external", if the interface provides a service that is used by other applications.

### References

Document Name	Version/Date	Location (Pages, Section)
Software Communications Architecture Specification	Version 2.2.2 05-05-2006	Page 3-96, 3-12

### Test Objective

This test case verifies AP0616. The objective of this test is to verify that all application interfaces providing service to other applications are properly defined in the Software Assembly Descriptor (SAD) file and declared as “external”.

### Places to Verify

Applications or any other component that provides services.

### IDL References

None.

### Preconditions

- The application source code files are available.
- The application design documents must be available.

## Test Description

- A. Determine whether the application provides services used by other applications via interfaces. (AP0616)
  - 1. **N/A:** The application does not provide services used by other applications via its interfaces (external). End test.
- B. Verify that external interfaces defined in the Software Design Document (SDD) are also defined in the *externalports* element of the SAD. (AP0616)
  - 1. **Pass:** The external interfaces defined in the Software Design Document are defined as “external” ports in the SAD xml file.
  - 2. **Fail:** The external interfaces defined in the Software Design Document are not defined as “external” ports in the SAD xml 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.

APP_TC_005				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine whether the application provides services used by other applications via interfaces. (AP0616)</b>				
1. Examine the software design document (SDD) to identify the requirements for external interfaces or HCI requirements. Alternatively, you may need to consult a developer to identify the requirements for external interfaces or HCI requirements.	<b>Pass:</b> The application provides services used by other applications via external interfaces. (AP0616)  <b>N/A:</b> The application does not provide any services via external interfaces. (AP0616)			
2. Record the external interfaces	External interfaces are found.		Use Log Sheet to write down the external interfaces.	
<b>B. Verify that external interfaces defined in the Software Design Document (SDD) are also defined in the <i>externalports</i> element of the SAD. (AP0616)</b>				
3. Determine the location of the SAD file.	Must have a SAD file.		Use the Log Sheet to write down the SAD file name.	
4. For every external interface recorded in step 2, verify that the <i>external interfaces</i> are identified as external ports.	<b>Pass:</b> The external <i>interfaces</i> are identified as external ports. (AP0616)  <b>Fail:</b> The external <i>interfaces</i> are not identified as external ports. (AP0616)		External ports are identified in the SAD as follows:  <pre> &lt;externalports&gt;   &lt;port&gt;     &lt;providesidentifier&gt;iiii   &lt;/providesidentifier&gt;   &lt;componentinstantiationref     refid="rrrr"&gt;   &lt;/componentinstantiationref&gt;   &lt;/port&gt; &lt;/externalports&gt; </pre>	
<b>End of Test</b>				



Test Recording Log – APP_TC_005	
SAD file:	
Step 2	Step 4
External interface	External ports exist.

**Test Summary APP\_TC\_005**

Once testing is complete for every component of the application 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):**

AP0616\_\_\_\_\_

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

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

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

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

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

## B.6 APP\_TC\_006 – ResourceFactory::releaseResource()

### Test Case Number: APP\_TC\_006

ResourceFactory::releaseResource()

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0116	The <i>releaseResource</i> operation shall decrement the reference count for the specified resource, as indicated by the <i>resourceId</i> parameter.
AP0117	The <i>releaseResource</i> operation shall release the resource from the CORBA environment and make the resource no longer available when the resource's reference count is zero.
AP0118	The <i>releaseResource</i> operation shall raise the <i>InvalidResourceId</i> exception if an invalid <i>resourceId</i> is received.

### References

Document Name	Version/Date	Location (Pages, Sections)
SCA	Version 2.2.2 05-15-2006	Page 3-18, Section 3.1.3.1.7.5.2.3 Page 3-19, Section 3.1.3.1.7.5.2.5 Page 3-23
SCA AppendixD	Version 2.2.2 05-15-2006	Page D-36 through D-38, Section D.6.1.3.3

### Test Objective

This test case verifies AP0116, AP0117 and AP0118. The objective of this test is to verify that the application's *ResourceFactory::releaseResource* operation will properly release a resource and remove it from the CORBA environment. This test also verifies that the *InvalidResourceId* exception is raised by *ResourceFactory::releaseResource* operation if an invalid *resourceId* is received. This test case is optional and will yield "N/A" if Resource Factory is not implemented. In CORBA there is client side and server side representation of a resource. The *releaseResource* operation provides the mechanism of releasing the resource in the CORBA environment on the server side when all clients are through with a specific resource.

### Places to Verify

Resource Factories

## IDL References

### Exceptions

exception InvalidResourceId{ };

### Operations

void releaseResource (in string resourceId) raises  
    {InvalidResourceId};

### Preconditions

- The application source code files are available.

### Test Description

A. Determine if the resource factories are used in the application. (AP0116, AP0117, AP0118)

1. **N/A:** The application does not implement a Resource Factory.

For each resource factory found, perform the following steps:

B. Verify that the releaseResource operation decrements the reference count for the specified resource. (AP0116)

1. **Pass:** The reference count is decremented.
2. **Fail:** The reference count is not decremented.

C. Verify that the releaseResource operation releases the resource from the CORBA environment when the reference count is zero. (AP0117)

1. **Pass:** The releaseResource operation releases the resource when the reference count is zero.
2. **Fail:** The releaseResource operation does not release the resource when the reference count is zero.

D. Verify that the releaseResource operation raises the exception InvalidResourceId when an invalid resourceId is received. (AP0118)

1. **Pass:** *ReleaseResource* raises the InvalidResourceId exception when an invalid resourceId is received.
2. **Fail:** *ReleaseResource* does not raise an InvalidResourceId exception when an invalid resourceId is received.



## 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.

APP_TC_006				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine if resource factories are implemented in the application. (AP0116, AP0117, AP0118)</b>				
1. Examine the SAD file declarations for evidence of ResourceFactory. List the SAD file and ResourceFactories.	N/A: The application does not implement a Resource Factory. (AP0116, AP0117, AP0118)		Example of the SAD string:  <findcomponent> <componentresourcefactoryref refid="DCE:12345678-1003-1000-8000-00A0C9E780D8">	
2. Identify the source code for the ResourceFactory. List the source file.	N/A: Source code for the ResourceFactory is not found. (AP0116)			
<b>For each resource factory found, perform the following steps:</b>				
<b>B. Verify that the releaseResource operation decrements the reference count for the specified resource. (AP0116)</b>				
3. Examine the source code and search for the portion of the releaseResource operation that decrements the reference count.	The reference count variable is found.		Example: virtual void releaseResource ( const CORBA::Char* resourceId, CORBA::Environment& _env ) = 0;	
4. Verify that the releaseResource operation decrements the reference count for a specified resource.	<b>Pass:</b> The reference count is decremented. (AP0116)  <b>Fail:</b> The reference count is not decremented. (AP0116)		The reference count is used to indicate the number of times that a specific resource reference has been given to requesting clients.	
<b>C. Verify that the releaseResource operation releases the resource from the CORBA environment when the reference count is zero. (AP0117)</b>				

APP_TC_006				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify that there is a check in the code for when the reference count is zero.	The code is located for the release when the count is zero.		The resource factory should not release a resource that has a reference count greater than zero.	
6. Verify that the releaseResource operation performs a shutdown process on the resource when the reference count is zero.	<p><b>Pass:</b> The releaseResource operation performs a shutdown on the resource when the reference count is zero. (AP0117)</p> <p><b>Fail:</b> The releaseResource operation does not perform a shutdown on the resource when the reference count is zero. (AP0117)</p>		A shutdown process should be initiated which includes the release of the resource from the CORBA environment	
<b>D. Verify that the releaseResource operation raises the exception InvalidResourceId when an invalid resourceId is received. (AP0118)</b>				
7. Find where the releaseResource operation verifies the input parameter resourceId.	The input string parameter is named resourceId or something similarly named.		"	
8. Verify that the releaseResource operation raises the exception InvalidResourceId when an invalid resourceId is received.	<p><b>Pass:</b> <i>ReleaseResource</i> raises the InvalidResourceId exception when an invalid resourceId is received. (AP0118)</p> <p><b>Fail:</b> <i>ReleaseResource</i> does not raise the InvalidResourceId exception when an invalid resourceId is received. (AP0118)</p>			
<b>End of Test</b>				

Test Recording Log – APP_TC_006					
SAD File:					
Step 1 (Resource Factory)	Step 2 (source file with releaseResource operation)	Step 4 (reference count decrement – Y/N?)	Step 6 (shutdown process initiated – Y/N?)	Step 8 (exception raised – Y/N?)	Notes

**Test Summary APP\_TC\_006**

Once testing is complete for every component of the application 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):**

AP0116 \_\_\_\_\_

AP0117 \_\_\_\_\_

AP0118 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.7 APP\_TC\_007 – ResourceFactory::shutdown raises ShutdownFailure

### Test Case Number: APP\_TC\_007

ResourceFactory::shutdown raises ShutdownFailure

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0107	The <i>shutdown</i> operation shall raise the ShutdownFailure exception when processing errors prevent the release of the resource factory from the CORBA environment or when all resources have not been released from the resource factory.
AP0119	The <i>shutdown</i> operation shall release the resource factory from the CORBA environment and make it unavailable to any subsequent calls to its object reference.

### References

Document Name	Version/Date	Location (Page, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-18, Section 3.1.3.1.7.3.2 Page 3-19 to 3-20, Section 3.1.3.1.7.5.3
SCA AppendixD	Version 2.2.2 05-15-2006	C-10 thru C-11

### Test Objective

This test case verifies AP0119 and AP0107. The objective of this test is to verify that the ResourceFactory::shutdown operation will properly release the Resource Factory from the CORBA environment. This test case will also verify that the ResourceFactory::shutdown operation will raise the ResourceFactory:: ShutdownFailure exception when errors prevent the release of the resource factory from CORBA or when resources have not been released from the resource factory. This test case is optional and will yield "N/A" if Resource Factory is not implemented.

### Places to Verify

ResourceFactories.

## IDL References

### Exception

exception ShutdownFailure { string msg; };

### Operation

void shutdown () raises (CF::ResourceFactory::ShutdownFailure); }

## Preconditions

- The application source code files are available.
- The SAD file is available.

## Test Description

A. Determine if the resource factories are implemented in the application. (AP0119, AP0107)

1. **N/A:** The application does not implement a Resource Factory.

For each resource factory found, perform the following steps:

B. Verify that the *shutdown* operation shall release the resource factory from the CORBA environment and make it unavailable to any subsequent calls to an object via its object reference. (AP0119)

1. **Pass:** The *shutdown* operation releases the resource factory from the CORBA environment and makes it unavailable to any subsequent calls to an object via its object reference.
2. **Fail:** The *shutdown* operation does not release the resource factory from the CORBA environment.
3. **Fail:** The *shutdown* operation makes the resource factory available to any subsequent calls to an object via its object reference.

C. Verify that *shutdown* operation shall raise the ShutdownFailure exception when processing errors prevent the release of the resource factory from the CORBA environment or when all resources have not been released from the resource factory. (AP0107)

1. **Pass:** A shutdown exception is raised for shutdown processing errors.
2. **Fail:** A shutdown exception is not raised for shutdown processing errors.

## 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.

APP_TC_007				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Determine if resource factories are implemented in the application. (AP0119, AP0107)</b>				
1. Examine the SAD file declarations for evidence of ResourceFactory. List the SAD file and ResourceFactories.	N/A: The application does not implement a Resource Factory. (AP0119, AP0107)		Example of the SAD string:  <findcomponent> <componentresourcefactoryref refid="DCE:12345678-1003-1000-8000-00A0C9E780D8">	
2. Search for the shutdown operation in the source files provided by the developer. Record the source file name.	<b>Fail:</b> The shutdown operation is not found in the source files. (AP0119, AP0107)			
<b>For each resource factory found, perform the following steps :</b>				
<b>B. Verify that the <i>shutdown</i> operation shall release the resource factory from the CORBA environment and make it unavailable to any subsequent calls to an object via its object reference. (AP0119)</b>				

APP_TC_007				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify that the shutdown operation releases the resource factory from the CORBA environment and make the resource factory unavailable to any subsequent calls to an object via its object reference.	<p><b>Pass:</b> The <i>shutdown</i> operation releases the resource factory from the CORBA environment and makes it unavailable to any subsequent calls to an object via its object reference. (AP0119)</p> <p><b>Fail:</b> The <i>shutdown</i> operation does not release the resource factory from the CORBA environment. (AP0119)</p> <p><b>Fail:</b> The <i>shutdown</i> operation makes the resource factory available to any subsequent calls to an object via its object reference. (AP0119)</p>		Examples of commands that may release objects from the CORBA environment 'release', 'destroy', 'deactivate_object',	
<b>C. Verify that <i>shutdown</i> operation shall raise the ShutdownFailure exception when processing errors prevent the release of the resource factory from the CORBA environment or when all resources have not been released from the resource factory. (AP0107)</b>				
4. Verify that the shutdown operation raises the ShutdownFailure exception when processing errors occur.	<p><b>Pass:</b> A <i>ShutdownFailure</i> exception is raised for shutdown processing errors. (AP0107)</p> <p><b>Fail:</b> A <i>ShutdownFailure</i> exception is not raised for shutdown processing errors. (AP0107)</p>		<p>Example:</p> <p>CF_CATCH(CF::ResourceFactory::ShutdownFailure, ex)</p>	



APP_TC_007				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify that the <i>shutdown operation</i> raises the ShutdownFailure exception when all resources have not been released.	<b>Pass:</b> A ShutdownFailure exception is raised when all resources have not been released. (AP0107)  <b>Fail:</b> A ShutdownFailure exception is not raised when all resources have not been released. (AP0107)			
End of Test				

Test Recording Log – APP_TC_007			
SAD File:			
Resource Factories:			
Step 2 (source files having shutdown operation)	Step 3 (shutdown release resource factories- Y/N?)	Step 4 (ShutdownFailure raised because of process errors- Y/N?)	Step 4 (ShutdownFailure raised because of unreleased resources- Y/N?)

## Test Summary APP\_TC\_007

Once testing is complete for every component of the application 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):**

AP0119 \_\_\_\_\_

AP0107 \_\_\_\_\_

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

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

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

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

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

## B.8 APP\_TC\_008 – AEP Mandatory OS Services

**Test Case Number:** APP\_TC\_008

AEP Mandatory OS Services

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0603	Applications shall be limited to using the OS services that are designated.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-95, Section 3.2.1.1
SCA Appendix B (Application Environment Profile)	Version 2.2.2 05-15-2006	Page B-1 to B-18

### Test Objective

This test case verifies AP0603. The objective of the test is to verify that the application is limited to using services that are designated mandatory in the SCA Application Environment Profile (AEP, SCA v2.2.2 Appendix B). This test will identify AEP OS service violations in the source code.

### Places to Verify

Applications and any other software component using OS services.

### IDL References

None

### Preconditions

- The application source code files are available.

### Test Description

Note: The best way to verify AEP compliance is to prove that violations to the requirement AP0603 do not exist. Below is a table of AEP OS service violations as compiled by JTEL.

For each component, perform the following:

A. Identify the source code files that use POSIX OS service functions. (AP0603)

1. **N/A:** The source code files are not available.

For each entry in the APP\_TC\_008 Table 1:

B. Search the source code for occurrences of that entry. (AP0603)

1. **Pass:** There were no occurrences of the table entry. Move on to the next table entry.

C. Research occurrences within the source code to verify that there is no usage of a POSIX OS service interface. (AP0603)

1. **Pass:** All occurrences represent other than usage of POSIX interfaces...
2. **Fail:** Some occurrence represents the usage of a POSIX interface.

The following is a table of AEP POSIX OS service calls that would violate requirement AP0603. They are to be used to perform steps B and C above.

APP_TC_008 Table 1			
List of violations of AEP Requirements			
_Exit	flockfile	memcpy	sched_get_priority_max
_exit	floorf	memcpy	sched_get_priority_min
_longjmp	floorl	memmove	sched_getparam
_setjmp	fma	memset	sched_getscheduler
_tolower	fmaf	mkfifo	sched_rr_get_interval
_toupper	final	mknod	sched_setparam
a64l	fmax	mksnprintf	sched_setscheduler
acosf	fmaxf	mktemp	sched_yield
acosh	fmaxl	mmap	seed48
acoshf	fmin	modff	seekdir
acoshl	fminf	modfl	sem_timedwait
acosl	fminl	mprotect	semctl
alarm	fmodf	mq_timedreceive	semget
asinf	fmodl	mq_timedsend	semop
asinh	fntmsg	mrnd48	sendmsg
asinhf	fnmatch	msgctl	setcontext
asinh1	fork	msgget	setegid
asinl	fpclassify	msgrcv	setenv

APP_TC_008 Table 1			
List of violations of AEP Requirements			
assert	fputwc	msgsnd	seteuid
atan2f	fputws	msync	setgid
atan2l	freeaddrinfo	munmap	setgrent
atanf	frexpf	nan	sethostent
atanh	frexpl	nanf	setitimer
atanhf	fsetpos	nanl	setkey
atanhl	fstatvfs	nearbyint	setlogmask
atanl	fsync	nearbyintf	setnetent
atexit	ftime	nearbyintl	setpgrp
atoll	ftok	nextafter	setpriority
basename	ftruncate	nextafterf	setprotoent
bcmp	ftrylockfile	nextafterl	setpwent
bcopy	ftw	nexttoward	setregid
bsd_signal	funlockfile	nexttowardf	setreuid
btowc	fwide	nexttowardl	setrlimit
bzero	fwprintf	nftw	setservent
cabs	fwscanf	nice	setsid
cabsf	gai_strerror	nl_langinfo	setstate
cabsl	gcvrt	rand48	setuid
cacos	getaddrinfo	openlog	setutxent
cacosf	getc_unlocked	optarg	shm_open
cacosh	getchar_unlocked	opterr	shm_unlink
cacoshf	getcontext	optind	shmat
cacoshl	getdate	optopt	shmctl
cacosl	getdate_err	pclose	shmdt
carg	getegid	pipe	shmget
cargf	getenv	poll	shutdown
cargl	geteuid	popen	sigaltstack
casin	getgid	posix_fadvise	sighold
casinf	getgrent	posix_fallocate	sigignore
casinh	getgrgid	posix_madvise	siginterrupt
casinhf	getgrgid_r	posix_mem_offset	siglongjmp
casinhl	getgrnam	posix_memalign	signbit

APP_TC_008 Table 1			
List of violations of AEP Requirements			
casinl	getgrnam_r	posix_openpt	siggam
catan	getgroups	posix_spawn	sigpause
catanf	gethostbyaddr	posix_spawn_file_actions_addclose	sigrelse
catanh	gethostbyname	posix_spawn_file_actions_adddup2	sigset
catanhf	gethostent	posix_spawn_file_actions_addopen	sigsetjmp
catanhl	gethostid	posix_spawn_file_actions_destroy	sinf
catanl	gethostname	posix_spawn_file_actions_init	sinhf
catclose	getitimer	posix_spawnattr_destroy	sinhl
catgets	getlogin	posix_spawnattr_getflags	sinl
catopen	getlogin_r	posix_spawnattr_getpgroup	sleep
cbrt	getmsg	posix_spawnattr_getschedparam	snprintf
cbrtf	getnameinfo	posix_spawnattr_getschedpolicy	socketmark
cbrtl	getnetbyaddr	posix_spawnattr_getsigdefault	socketpair
ccos	getnetbyname	posix_spawnattr_getsigmask	sqrtf
ccosf	getnetent	posix_spawnattr_init	sqrtl
ccosh	getopt	posix_spawnattr_setflags	srand48
ccoshf	getpeername	posix_spawnattr_setpgroup	srandom
ccoshl	getpgid	posix_spawnattr_setschedparam	statvfs
ccosl	getpgrp	posix_spawnattr_setschedpolicy	strcasecmp
ceilf	getpid	posix_spawnattr_setsigdefault	strcoll
ceill	getpmsg	posix_spawnattr_setsigmask	strdup
cexp	getppid	posix_spawnnp	strerror
cexpf	getpriority	posix_trace_attr_destroy	strerror_r
cexpl	getprotobyname	posix_trace_attr_getclockres	strfmon
cfgetispeed	getprotobynumber	posix_trace_attr_getcreatetime	strncasecmp
cfgetospeed	getprotoent	posix_trace_attr_getgenversion	strptime
cfsetispeed	getpwent	posix_trace_attr_getinherited	strtod
cfsetospeed	getpwnam	posix_trace_attr_getlogfullpolicy	strtof
chmod	getpwnam_r	posix_trace_attr_getlogsize	strtoimax
chown	getpwuid	posix_trace_attr_getmaxdatasize	strtol
cimag	getpwuid_r	posix_trace_attr_getmaxsystemeventsizesize	strtold
cimagf	getrlimit	posix_trace_attr_getmaxuserereventsizesize	strtoll
cimagl	getrusage	posix_trace_attr_getname	strtoul

APP_TC_008 Table 1			
List of violations of AEP Requirements			
clock	gets	posix_trace_attr_getstreamfullpolicy	strtoull
clock_getcpuclockid	getservbyname	posix_trace_attr_getstreamsize	strtoimax
clock_nanosleep	getservbyport	posix_trace_attr_init	strxfrm
clog	getservent	posix_trace_attr_setinherited	swab
clogf	getsid	posix_trace_attr_setlogfullpolicy	swapcontext
clogl	getsockname	posix_trace_attr_setlogsize	swprintf
closelog	getsubopt	posix_trace_attr_setmaxdatasize	swscanf
confstr	gettimeofday	posix_trace_attr_setname	symlink
conj	getuid	posix_trace_attr_setstreamfullpolicy	sync
conf	getutxent	posix_trace_attr_setstreamsize	SynchronousReceive
conjl	getutxid	posix_trace_clear	SynchronousSend
copysign	getutxline	posix_trace_close	sysconf
copysignf	getwc	posix_trace_create	syslog
copysignl	getwchar	posix_trace_create_withlog	system
cosf	getwd	posix_trace_event	tanf
coshf	glob	posix_trace_eventid_equal	tanhf
coshl	globfree	posix_trace_eventid_get_name	tanhf
cosl	grantpt	posix_trace_eventid_open	tanl
cpow	HaltTask	posix_trace_eventset_add	tcdrain
cpowf	hcreate	posix_trace_eventset_del	tcflow
cpowl	hdestroy	posix_trace_eventset_empty	tcflush
cproj	hsearch	posix_trace_eventset_fill	tcgetattr
cprojf	hypot	posix_trace_eventset_ismember	tcgetpgrp
cprojl	hypotf	posix_trace_eventtypelist_getnext_id	tcgetsid
creal	hypotl	posix_trace_eventtypelist_rewind	tcsendbreak
crealf	iconv	posix_trace_flush	tcsetattr
creall	iconv_close	posix_trace_get_attr	tcsetpgrp
crypt	iconv_open	posix_trace_get_filter	tdelete
csin	if_freenameindex	posix_trace_get_status	telldir
csinf	if_indexoname	posix_trace_getnext_event	tempnam
csinh	if_nameindex	posix_trace_open	tfind
csinhf	if_nametoindex	posix_trace_rewind	tgamma
csinhf	ilogb	posix_trace_set_filter	tgammaf



APP_TC_008 Table 1			
List of violations of AEP Requirements			
csinl	ilogbf	posix_trace_shutdown	tgamma
csqrt	ilogbl	posix_trace_start	TimedSynchronousReceive
csqrtf	imaxabs	posix_trace_stop	TimedSynchronousSend
csqrtl	imaxdiv	posix_trace_timedgetnext_event	times
ctan	index	posix_trace_trid_eventid_open	timezone
ctanf	inet_addr	posix_trace_trygetnext_event	toascii
ctanh	inet_ntoa	posix_typed_mem_get_info	towctrans
ctanhf	inet_ntop	posix_typed_mem_open	towlower
ctanhl	inet_pton	powf	towupper
ctanl	initstate	powl	trunc
ctermid	insque	pread	truncate
CurrentTask	ioctl	pselect	truncf
daylight	isascii	pthread_atfork	truncl
dbm_clearerr	isastream	pthread_barrier_destroy	tsearch
dbm_close	isatty	pthread_barrier_init	ttynname
dbm_delete	isblank	pthread_barrier_wait	ttynname_r
dbm_error	isfinite	pthread_barrierattr_destroy	twalk
dbm_fetch	isgreater	pthread_barrierattr_getpshared	tzname
dbm_firstkey	isgreaterequal	pthread_barrierattr_init	tzset
dbm_nextkey	isinf	pthread_barrierattr_setpshared	ualarm
dbm_open	isless	pthread_condattr_getclock	ulimit
dbm_store	islessequal	pthread_condattr_getpshared	umask
difftime	islessgreater	pthread_condattr_setclock	uname
dimame	isnan	pthread_condattr_setpshared	ungetwc
div	isnormal	pthread_getconcurrency	unlockpt
dlclose	isunordered	pthread_getcpuclockid	unsetenv
dlderror	iswalnum	pthread_mutex_timedlock	usleep
dlopen	iswalph	pthread_mutexattr_getpshared	utimes
dlsym	iswblank	pthread_mutexattr_gettype	va_arg
drand48	iswcntrl	pthread_mutexattr_setpshared	va_copy
dup	iswctype	pthread_mutexattr_settype	va_end
dup2	iswdigit	pthread_rwlock_destroy	va_start
ecvt	iswgraph	pthread_rwlock_init	vfork

APP_TC_008 Table 1			
List of violations of AEP Requirements			
encrypt	iswlower	pthread_rwlock_rdlock	vfprintf
endgrent	iswprint	pthread_rwlock_timedrdlock	vfsconf
endhostent	iswpunct	pthread_rwlock_timedwrlock	vwfprintf
endnetent	iswspace	pthread_rwlock_tryrdlock	vwscanf
endprotoent	iswupper	pthread_rwlock_trywrlock	vprintf
endpwent	iswxdigit	pthread_rwlock_unlock	vscanf
endservent	j0	pthread_rwlock_wrlock	vsnprintf
endutxent	j1	pthread_rwlockattr_destroy	vsprintf
environ	jn	pthread_rwlockattr_getpshared	vsscanf
erand48	jrand48	pthread_rwlockattr_init	vswprintf
erf	killpg	pthread_rwlockattr_setpshared	vswscanf
erfc	l64a	pthread_setconcurrency	vwprintf
erfcf	labs	pthread_setschedprio	vwscanf
erfcl	lchown	pthread_spin_destroy	wait
erff	lcong48	pthread_spin_init	waitid
erfl	ldexpf	pthread_spin_lock	waitpid
execl	ldexpl	pthread_spin_trylock	wcrtomb
execle	ldiv	pthread_spin_unlock	wscat
execlp	lfind	ptsname	wcschr
execv	lgamma	putc_unlocked	wscmp
execve	lgammaf	putchar_unlocked	wscoll
execvp	lgammal	putenv	wscpy
exit	llabs	putmsg	wscspn
Exit	lldiv	putpmsg	wcsftime
exp2	llrint	pututxline	wcslen
exp2f	llrintf	putwc	wcsncat
exp2l	llrintl	putwchar	wcsncmp
expf	llround	pwrite	wcsncpy
expl	llroundf	random	wcsprk
expml	llroundl	readlink	wcsrchr
expm1f	localeconv	readv	wcsrtombs
expm1l	lockf	realpath	wcsspn
fabsf	log10f	recvmsg	wcsstr

APP_TC_008 Table 1			
List of violations of AEP Requirements			
fabsl	log10l	regcomp	wcstod
fattach	log1p	regerror	wcstof
fchdir	log1pf	regexec	wcstimax
fchmod	log1pl	regfree	wcstok
fchown	log2	ReleaseResource	wcstol
fcntl	log2f	remainder	wcstold
fcvt	log2l	remainderf	wcstoll
FD_CLR	logb	remainderl	wcstombs
FD_ISSET	logbf	remque	wcstoul
FD_SET	logbl	remquo	wcstoull
FD_ZERO	logf	remquof	wcstoumax
fdatasync	logl	remquol	wcs wcs
fdetach	lrnd48	RequestResource	wcs width
fdim	lrint	rindex	wcs xfrm
fdimf	lrintf	rint	wctob
fdiml	lrintl	rintf	wctomb
feclearexcept	lround	rintl	wctrans
fegetenv	lroundf	round	wctype
fegetexceptflag	lroundl	roundf	wcwidth
fegetround	lsearch	roundl	wmemchr
feholdexcept	lstat	scalb	wmemcmp
feraiseexcept	makecontext	scalbln	wmemcpy
fesetenv	mblen	scalblnf	wmemmove
fesetexceptflag	mbrlen	scalblnl	wmemset
fesetround	mbrtowc	scalbn	wordexp
fetestexcept	mbsinit	scalbnf	wordfree
feupdateenv	mbsrtowcs	scalbnl	wprintf
ffs	mbstowcs		wrtv
fgetpos	mbtowc		wscanf
fgetwc	memccpy		y0
fgetws	memchr		y1
			yn

## 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.

APP_TC_008				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>For each component, perform the following:</b>				
<b>A. Identify the source code files that use POSIX OS service functions. (AP0603)</b>				
1. Perform a search for all OS service functions in the application source code provided by the developer. Record the source file name(s).	N/A: The source code files are not available. (AP0603)			
<b>For each entry in the APP_TC_008 Table 1 perform steps 2 &amp; 3 :</b>				
<b>B. Search the source code for occurrences of that entry. (AP0603)</b>				
2. Using the select entry from Table 1, perform a search for that entry.	<b>Pass:</b> There are no occurrences of the table entry. Move on to the next table entry. (AP0603)  Otherwise go to Step 3			
<b>C. Research occurrences within the source code to verify that there is no usage of a POSIX interface. (AP0603)</b>				
3. Research occurrences within the source code to verify that there is no usage of a POSIX interface.	<b>Pass:</b> All occurrences of the table entry found in the search do not represent usage of a POSIX interface. (AP0603)  <b>Fail:</b> One or more occurrences of the table entry found in the search, represents usage of a POSIX interface. (AP0603)			
<b>End of Test</b>				

Test Recording Log – APP_TC_008		
Step 1 (source file having OS service function calls)	Step 2 (table entries that are present)	Step 3 (Result of research of AEP OS violation)

## Test Summary APP\_TC\_008

Once testing is complete for every component of the application under test, report for each requirement 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):**

AP0603 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.9 APP\_TC\_009 – TestableObject::runTest() Properties

**Test Case Number:** APP\_TC\_009

*TestableObject::runTest*

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0082	Valid testId(s) and both input and output testValues (properties) for the <i>runTest</i> operation shall at a minimum be the test properties defined in the properties test element of the component's Properties Descriptor (refer to AppendixD Domain Profile).
AP0083	All testValues parameter properties (i.e., test properties defined in the propertyfile(s) referenced in the component's SPD) shall be validated.

### References

Document Name	Version/Date	Location (Pages,Section)
SCA	Version 2.2.2 05-15-2006	Page 3-10 – 3-11, Section 3.1.3.1.3.5.1.3
SCA AppendixD	Version 2.2. 05-15-2006	Page D4-D13, Section D.2 Page D19-D32, Section D.4 and D.5

### Test Objective

This test case verifies AP0082 and AP0083. The objective of this test case is to verify that the *runTest()* operation will accept any test properties listed in the domain profile files. It also verifies that the input values are validated before they are used.

### Places to Verify

TestableObject

### IDL References

#### Operation

```
void runTest ( in unsigned long testid, inout CF::Properties testValues )  
    raises (CF::TestableObject::UnknownTest, CF::UnknownProperties);
```

### Preconditions

- The application source code files are available.

## Test Description

- A. Identify the *testId*'s, *inputvalue*'s and *resultvalue*'s from the PRF files.
  - 1. **N/A:** No *testId*'s are found in the PRF files. End the test.
  - 2. **N/A:** No *inputvalue*'s or *resultvalue*'s are found. (skip steps C and D)
- B. Verify that the *runTest* operation accepts the id of the *testValues*'s identified in the PRF file (AP0082).
  - 1. **Fail:** An id of the *testValues*'s is not accepted
  - 2. **Pass:** The input id of the *testValues*'s are accepted (AP0082).
- C. Verify that the *value* of the *testValues* is validated (AP0083).
  - 1. **Pass:** The input *testValues* are validated.
  - 2. **Fail:** The input *testValues* are not validated.
- D. Verify that the *runTest* operation outputs the *testId*'s identified in the PRF file (AP0082).
  - 1. **Fail:** *runTest* does not return the set of output *testValues*.
  - 2. **Pass:** *RunTest* does return the set of output *testValues*.



## 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.

APP_TC_009				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the <i>testId</i>'s from the PRF files.</b>				
1. Use the SAD file to find the application's SPD files.	A list of 1 or more SPD files.			
2. For each SPD file determine its SCD and PRF files.	A list of SCD and PRF files.		Identifying the component that each PRF file is associated with will help in subsequent steps.	
3. For each SCD file determine its PRF files and add it to the list of PRF files.	PRF files are found.		If the list of PRF files is empty, then mark all requirements as N/A and end the test.	
4. Search the PRF for <i>testIds</i> and <i>testValues</i> and record the <i>testId</i> and any <i>inputvalue</i> 's and <i>resultvalue</i> 's associated with it. For the <i>inputvalue</i> and the <i>resultvalue</i> record the <i>id</i> and <i>value</i> .	Pass: <i>TestId</i> 's are found.  N/A: If there are no such elements, then end the test.  Pass: <i>inputvalue</i> 's and <i>resultvalue</i> 's are found.		In this example of the XML, the bolded fields are what need to be recorded. The <i>kindtype</i> must be "test":  <pre> &lt;test id="SOME_TEST"&gt;   &lt;description&gt;SOME_TEST &lt;/description&gt;   &lt;inputvalue&gt;     &lt;simple id="TEST_VALUE"       type="ulong"       name="SOME_TEST_INPUT"       mode="readonly"&gt;       &lt;description&gt;input test value       &lt;/description&gt;       &lt;value&gt;1000&lt;/value&gt;       &lt;kind kindtype="test"/&gt;     &lt;/simple&gt;   &lt;/inputvalue&gt;   &lt;resultvalue&gt;     &lt;simple       id="Some_TEST_RETURN_VALUE"       type="ulong"           </pre>	

APP_TC_009				
Steps	Expected Results	Actual Results	Comments	Test Result
			<pre> name="SOME_TEST_RETURN_VALUE" mode="readonly"&gt;   &lt;description&gt;Expected result&lt;/description&gt;   &lt;value&gt;0&lt;/value&gt; &lt;/simple&gt; &lt;/resultvalue&gt; &lt;/test&gt; </pre>	
<p>Note: <i>runTest</i> looks like:</p> <pre>void runTest (in unsigned long testId, inout Properties testValues) raises (UnknownTest, UnknownProperties);</pre> <p>The <i>id</i> and <i>value</i> recorded in step 4 form a properties pair. The <i>testValues</i> input to <i>runTest</i> is a set of these pairs.</p>				
<b>B. Verify that the <i>runTest</i> operation accepts the id of the <i>testValues</i>'s identified in the PRF file (AP0082).</b>				
5. Locate the <i>runTest</i> code in the application source code files and record the file names.	<p><b>Pass:</b> Code is located. (AP0082)</p> <p><b>Fail:</b> No <i>runTest</i> code exists. (AP0082)</p>		There are <i>testId</i> 's in the xml but no <i>runTest</i> code to process them, so the requirements fail.	
6. Verify that the source code accepts valid id of the <i>testValues</i> 's.	<p><b>Pass:</b> The id's listed in step 4 are accepted. (AP0082)</p> <p><b>Fail:</b> A <i>testId</i> is rejected. (AP0082)</p>			
<b>C. Verify that the values of the testValues are validated (AP0083).</b>				
7. Verify that the values of the input <i>testValues</i> 's are validated	<p><b>Pass:</b> Values of the input testValues are validated. (AP0083)</p> <p><b>Fail:</b> Values of the input testValues are not validated. (AP0083)</p>		.	
<b>D. Verify that the <i>runTest</i> operation outputs the <i>testId</i>'s identified in the PRF file (AP0082).</b>				

APP_TC_009				
Steps	Expected Results	Actual Results	Comments	Test Result
8. Verify that the <i>resultvalue</i> is set before <i>runTest</i> exits.	Pass: The value of <i>resultvalue</i> is set. (AP0082)  Fail: The value of <i>resultvalue</i> is not set. (AP0082)			
End of Test				

Test Recording Log – APP_TC_009							
SAD file:							
Step 1 (SPD files)	Step 2 and 3 (SCA and PRF files)	Step 4 (testId)	Step 4 (testValues – id and value pair)	Step 5 (runTest code files)	Step 6 (Id)	Step 7 (value)	Step (result value)

## Test Summary APP\_TC\_009

Once testing is complete for every component of the application 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):**

AP0082 \_\_\_\_\_

AP0083 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.10APP\_TC\_010 – TestableObject::runTest()

**Test Case Number:** APP\_TC\_010

TestableObject::runTest()

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0079	The <i>runTest</i> operation shall use the input testId parameter to determine which of its predefined test implementations should be performed
AP0080	The id/value pair(s) of the testValues parameter shall be used to provide additional information to the implementation-specific test to be run.
AP0081	The <i>runTest</i> operation shall return the result(s) of the test in the testValues parameter.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-10 thru 3-11, Section 3.1.3.1 Page 3-15, Section 3.1.3.6.2 Page 3-29, Section 3.1.3.2.2.5.1.3
SCA AppendixD – Domain Profile	Version2.2.2 05-15-2006	Page D-4, Section D.2.1 Page D-19 thru D-25, Section D.4 Page D-34, Section D.6.1.2

### Test Objective

This test case verifies AP0079, AP0080, and AP0081. The objective of this test is to verify that the *runTest()* operation has the functionality to test component implementations. Additionally, the *runTest* operation uses the testId to determine the predefined test implementations and testValues for additional information to the implementation-specific tests to be performed. Lastly, the test verifies that the test result(s) is/are returned via the testValues parameter by the *runTest* operation.

### Places to Verify

TestableObject

### IDL References

#### Operation

void runTest ( in unsigned long testId, inout CF::Properties testValues )

---

raises (CF::TestableObject::UnknownTest, CF::, CF::UnknownProperties);

## Preconditions

- The application source code files are available.

## Test Description

For each component:

- A. Verify that runTest() uses the input testId parameter to determine which of its tests should be performed. (AP0079)
  1. **Fail:** The testId is not used to determine the test it will run.
  2. **Pass:** The testId is used to determine the test it will run.
- B. Verify that runTest() uses the id/value pair(s) of the testValue parameter to provide additional information about the test to run. (AP0080)
  1. **Pass:** The testValue parameters provide additional information to the test.
  2. **Fail:** The testValue parameters do not provide additional information to the test.
- C. Verify that runTest() returns the results of the test in the testValue parameter. (AP0081)
  1. **Pass:** The testValue parameter contains the return value from runTest.
  2. **Fail:** The testValue parameter does not contain the return value from runTest.

## 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.

APP_TC_010				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>For each component:</b>				
<b>A. Verify that runTest() uses the input testId parameter to determine which of its tests should be performed. (AP0079)</b>				
1. Locate the source code for this component's runTest.	The source code is found.		The software developer is the best source for this information.	
2. Verify runTest(), using a valid testId, will execute the proper predefined test. (AP0079)	<b>Pass:</b> The testId is validated and is used to select the test that is executed. (AP0079)  <b>Fail:</b> The testId is not validated or used to select the test that is executed. (AP0079)		The synopsis of this operation:  <pre>void runTest (     in unsigned long testId,     inout Properties testValues) raises (UnknownTest,         UnknownProperties);</pre>	
3. Verify runTest() passes the input value to the test in the testValue parameter. (AP0079)	<b>Pass:</b> The testValue parameter contains the input value. (AP0079)  <b>Fail:</b> The testValue parameter contains the input value. (AP0079)			
<b>B. Verify that runTest() uses the id/value pair(s) of the testValue parameter to provide additional information about the test to run. (AP0080)</b>				
4. Verify that runTest() uses the id/value pair(s) of the testValue parameter to provide additional information about the test to run. (AP0080)	<b>Pass:</b> The testValue parameters provide additional information to the test. (AP0080)  <b>Fail:</b> The testValue parameters do not provide additional information to the test. (AP0080)			



APP_TC_010				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>C. Verify that runTest() returns the results of the test in the testValue parameter.(AP0081)</b>				
<b>5.</b> Verify runTest(), will return the return value in the testValue parameter. (AP0081)	<b>Pass:</b> The testValue parameter contains the return value from runTest. (AP0081)  <b>Fail:</b> The testValue parameter does not contain the return value from runTest. (AP0081)			
<b>End of Test</b>				

Test Recording Log – APP_TC_010				
Step 1 (source code found)	Step 2 (testId)	Step 3 (input value in testValue parameter)	Step 4 (testValue parameter provides additional information)	Step 5 (return value found in testValue parameter)

## Test Summary APP\_TC\_010

Once testing is complete for every component of the application 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):**

AP0079 \_\_\_\_\_

AP0080 \_\_\_\_\_

AP0081 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.11APP\_TC\_011 – PropertySet::configure()

**Test Case Number:** APP\_TC\_011

PropertySet::configure()

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0091	The <i>configure</i> operation shall assign values to the properties as indicated in the input <i>configProperties</i> parameter.
AP0092	Valid properties for the <i>configure</i> operation shall at a minimum be the <i>configure readwrite</i> and <i>writeonly</i> properties referenced in the component's SPD.
AP0093	The <i>configure</i> operation shall raise a <i>PartialConfiguration</i> exception when some configuration properties were successfully set and some configuration properties were not successfully set.
AP0094	The <i>configure</i> operation shall raise an <i>InvalidConfiguration</i> exception when a configuration error occurs and no configuration properties were successfully set.

### References

Document Name	Version/Date	Location (Pages,Section)
SCA	Version 2.2. 05-15-2006	Page 3-13 to 3-15
SCA AppendixD	Version 2.2.2 05-15-2006	Page D1-D13 and D19-D32

### Test Objective

This test case verifies AP0091, AP0092, AP0093 and AP0094. The objective of this test is to verify that the *configure()* operation is based on the list of *configure* properties defined in the PRF file(s) of the application under test. This test verifies that a component can set the values of *configure* properties with mode of type *readwrite* or *writeonly*. In addition, this test verifies that a component throws a *PartialConfiguration* exception when some configuration properties were successfully set and some were not. It ensures that a component throws an *InvalidConfiguration* exception when an error occurs and no configuration properties are set.

### Places to Verify

Resource or any other component that uses PropertySet interface to assign names and values to properties.

### IDL References

#### Exceptions

exception InvalidConfiguration {

```
string msg;  
CF::Properties invalidProperties; };  
exception PartialConfiguration {  
    CF::Properties invalidProperties; };
```

### Operations

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

### Preconditions

- The application source code files are available.

### Test Description

- A. Search PRF file(s) identified in SPD files and locate all the simple elements with a kind *kindtype="configure"* and a mode attribute set to either *"readwrite"* or *"writeonly"*. For each element found, store the id, type, value (if defined) and range (if defined) in a *configElements* list. Ignore duplicate *configure* properties, and do not insert them into the list.
1. **N/A:** No *readwrite* or *writeonly* configuration properties are defined. (AP0092)
- For each component of the Core Framework that contains configurable properties referenced in its SPD, perform the following steps:
- B. Verify that the *configure()* operation assigns the value associated with the id from the *configProperties* parameter to the value of the property with same id in the component and the value is stored. (AP0091)
1. **Pass:** The *configure()* operation assigns the value associated with the id of the *configProperties* parameter to the value of the property with the same id in the component and the value is stored.
  2. **Fail:** The *configure()* operation does not assign the value associated with the id of the *configProperties* parameter to the value of the property with the same id in the component or the value is not stored.
- C. Verify that a call to the *configure()* operation will set properties that are passed in via the *configProperties* parameter; if the properties are of type *readwrite* or *writeonly*. (AP0092)
1. **Pass:** Properties passed in as *configure()* operation parameters of the proper type are assigned.
  2. **Fail:** One or more valid properties are not handled properly (assigned) within the *configure()* operation.
- D. Verify that the *configure()* operation sets all parameters that are defined in the component's PRFs. There can be additional properties (i.e. unadvertised capabilities) that are supported but not fewer properties than those in the PRFs. (AP0092)

- E. If there is logic in the *configure()* operation that can result in the rejection of a property setting assignment (e.g. range checking done on inputs within the *configure()* operation), then verify the *configure()* operation has implemented a provision for the *InvalidConfiguration* exception to be raised. (AP0094)
1. **Pass:** If all of the *configure* property assignments are rejected then the *InvalidConfiguration* exception is raised.
  2. **Fail:** If all *configure* property assignments may be rejected, but the exception is never raised.
- F. If there are multiple properties within the component, and if there is logic in the *configure()* operation that can result in the rejection of a property setting assignment (e.g. range checking done on inputs within the *configure()* operation) on one or more of the properties, then verify the *configure()* operation has implemented a provision for the *PartialConfiguration* exception to be raised. (AP0093)
1. **Fail:** If some *configure()* property assignments are rejected, but the exception is never raised.
  2. **Pass:** If some of the *configure()* property assignments are rejected then the *PartialConfiguration* exception is 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.

APP_TC_011				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Search PRF file(s) identified in SPD files and locate all the simple elements with a kind kindtype="configure" and a mode attribute set to either "readwrite" or "writeonly". (AP0092)</b>				
1. Identify each PRF file.	PRF files identified		(list PRF files on log sheet)	
2. Identify each <i>readwrite</i> or <i>writeonly</i> attribute. Ignore duplicate <i>configure</i> properties, and do not include them in the list.	N/A: No <i>readwrite</i> or <i>writeonly</i> configuration properties are defined. (AP0092)		Find simple elements with a kind <i>kindtype="configure"</i> and a <i>mode</i> attribute set to either " <i>readwrite</i> " or " <i>writeonly</i> ". For each element found, store the id, type, value (if defined) and range (if defined) in a <i>configElements</i> list.	
<b>For each component of the Core Framework that contains configurable properties referenced in its SPD, perform the following steps:</b>				
<b>B. Verify that the <i>configure()</i> operation assigns the value associated with the id from the <i>configProperties</i> parameter to the value of the property with same id in the component and the value is stored. (AP0091)</b>				
3. For each component, find the source code for the <i>configure()</i> operation	Source code located			
4. Find where the <i>configure()</i> operation verifies the input parameter <i>configProperties</i> .	Parameter located			
5. Verify that the <i>configure()</i> operation iterates through the list of <i>configProperties</i> storing the property values based on the corresponding id.	<p><b>Pass:</b> Properties passed in as <i>configure()</i> operation parameters are assigned (AP0091)</p> <p><b>Fail:</b> One or more valid properties are not handled properly (assigned) within the <i>configure()</i> operation. (AP0091)</p>			

APP_TC_011				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>C. Verify that a call to the <i>configure()</i> operation will set properties that are passed in via the <i>configProperties</i> parameter; if the properties are of type <i>readwrite</i> or <i>writeonly</i>. (AP0092)</b>				
6. Verify that the <i>configure()</i> operation sets properties listed in the input parameter, if they are of type <i>readwrite</i> or <i>writeonly</i> .	<p><b>Pass:</b> Properties passed in as <i>configure()</i> operation parameters are assigned if they are of type <i>readwrite</i> or <i>writeonly</i>. (AP0092)</p> <p><b>Fail:</b> One or more valid properties are not handled properly (assigned) within the <i>configure()</i> operation. (AP0092)</p>		The iteration should work even if the <i>configProperties</i> list is supplied in order that is different from the order of their PRF appearance.	
<b>D. Verify that the <i>configure()</i> operation sets all parameters that are defined in the component's PRFs. (AP0092)</b>				
7. Verify that <i>readwrite</i> and <i>writeonly</i> attributes are handled by the <i>configure()</i> operation.	<p><b>Pass:</b> All declared <i>readwrite</i> and <i>writeonly</i> attributes with a kindtype of <i>configure</i> are handled by the <i>configure()</i> operation. (AP0092)</p> <p><b>Fail:</b> Not all declared <i>readwrite</i> and <i>writeonly</i> attributes with a kindtype of <i>configure</i> are handled by the <i>configure()</i> operation. (AP0092)</p>			
<b>E. If there is logic in the <i>configure()</i> operation that can result in the rejection of a property setting assignment (e.g. range checking done on inputs within the <i>configure()</i> operation), then verify the <i>configure()</i> operation has implemented a provision for the <i>InvalidConfiguration</i> exception to be raised. (AP0094)</b>				



APP_TC_011				
Steps	Expected Results	Actual Results	Comments	Test Result
8. Examine the implementation of the <i>configure()</i> operation and verify that it raises an <i>InvalidConfiguration</i> exception when a configuration error occurs and no configuration properties were successfully set.	<p><b>Pass:</b> Checking is performed, if all of the <i>configure</i> property assignments are rejected then the <i>InvalidConfiguration</i> exception is raised. (AP0094)</p> <p><b>Fail:</b> Checking is performed, all <i>configure</i> property assignments may be rejected, but the exception is never raised. (AP0094)</p>		If one or more attributes has range checking prior to assignment, then it will be possible to construct a scenario wherein all assignments are invalid.	
<b>F. If there are multiple properties within the component, and if there is logic in the <i>configure()</i> operation that can result in the rejection of a property setting assignment (e.g. range checking done on inputs within the <i>configure()</i> operation) on one or more of the properties, then verify the <i>configure()</i> operation has implemented a provision for the <i>PartialConfiguration</i> exception to be raised. (AP0093)</b>				
9. Examine the <i>configure()</i> operation and verify that the <i>PartialConfiguration</i> exception is thrown in the event where some properties assignments are rejected but others are accepted.	<p><b>Pass:</b> The <i>configure()</i> operation throws <i>PartialConfiguration</i> exception in the event where some properties assignments are rejected but others are accepted. (AP0093)</p> <p><b>Fail:</b> The <i>configure()</i> operation does not throw <i>PartialConfiguration</i> exception in the event where some properties assignments are rejected but others are accepted. (AP0093)</p>		If there is only one property, or if no properties have range checking prior to assignment, then it will not be possible to construct a partial configuration scenario.	
End of Test				

Test Recording Log – APP_TC_011					
SAD File:					
Step 1 & 2 Attributes in PRF (Y / N)	Step 5 & 6 Assigned by <i>configure()</i> (Y / N)	Step 6 <i>readwrite</i> <i>writeonly</i> (ignore readonly attributes)	Step 7 Allows the setting of parameters in PRFs (Y / N)	Step 8 Invalid Configuration raised (Y / N)	Step 9 Partial Configuration raised (Y / N)

## Test Summary APP\_TC\_011

Once testing is complete for every component of the application 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):**

AP0091 \_\_\_\_\_

AP0092 \_\_\_\_\_

AP0093 \_\_\_\_\_

AP0094 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.12APP\_TC\_012 – Port::disconnectPort raises InvalidPort

### Test Case Number: APP\_TC\_012

Port::disconnectPort

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0072	The <i>disconnectPort</i> operation shall break the connection to the component identified by the input <i>connectionId</i> parameter.
AP0073	The <i>disconnectPort</i> operation shall raise the <i>InvalidPort</i> exception when the input <i>connectionId</i> parameter is not a known connection to the <i>Port</i> component.
C003	The <i>InvalidPort</i> exception indicates one of the following errors has occurred in the specification of a Port association: 2. <i>errorCode</i> 2 means the Port name is not found (not used by this Port).

### References

Document Name	Version/Date	Location (page, section)
SCA	v2.2.2 05-15-2006	Pages 3-6 – 3-8, Section 3.1.3.1.1.5.2

### Test Objective

This test case verifies AP0072, AP0073 and associated criteria C003. The objective of this test case is to verify that the component shall break a specified connected port when the *disconnectPort* operation is executed. The test case also verifies that the *Invalid Port* exception is thrown when the *connectionId* parameter passed to the *disconnectPort* operation is not a known connection. The test case also verifies that the *errorCode* value in the raised exception is 2.

### Places to verify

Implementations of the Port interface.

### IDL Reference

#### Exception

```
exception InvalidPort { unsigned short errorCode;  
                        string msg; };
```

**Operation**

void disconnectPort ( in string connectionId )  
    raises (CF::Port::InvalidPort);

**Preconditions**

- The source code files for the Port Interface are available.

**Test Description**

A. Verify that the source code for the *Port disconnectPort* operation is found. (AP0072)

1. **Pass:** The source code for the *Port disconnectPort* operation is found.
2. **Fail:** The source code for the *Port disconnectPort* operation is not found.

For each instance of a *disconnectPort* operation found:

B. Verify that the *InvalidPort* exception is raised, when the input *connectionId* parameter is not a known connection to the Port component. (AP0073)

1. **Pass:** The *InvalidPort* exception is raised when the input *connectionId* parameter is not a known connection to the Port component.
2. **Fail:** The *InvalidPort* exception is not raised when the input *connectionId* parameter is not a known connection to the Port component.

C. Verify the presence of an error code value of 2 when the *Port* name is not found. (C003)

1. **Pass:** The *InvalidPort* exception has an error code value of 2 when the *Port* name is not found.
2. **Fail:** The *InvalidPort* exception does not have an error code value of 2 when the *Port* name is not found.

D. Verify that when no exception is raised, the port identified by *connectionId* is successfully disconnected using the *disconnectPort* operation.h (AP0072)

1. **Pass:** The port identified by *connectionId* is successfully disconnected using the *disconnectPort* operation.
2. **Fail:** The port identified by *connectionId* is not successfully disconnected using the *disconnectPort* operation.

## Manual Test Steps

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

2. The Test Recording Log is intended to record long comments and lists of SPD, source codes and other files/data.

APP_TC_012				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Verify that the source code for the <i>Port disconnectPort</i> operation is found. (AP0072, AP0073)</b>				
1. Identify and record the source code for the Port component.	<b>Pass:</b> The Port component is found. (AP0072, AP0073)			
2. Locate and record the file name of the source code for the <i>disconnectPort</i> operation of the Port component.	<b>Pass:</b> File is found. (AP0072, AP0073) <b>Fail:</b> File is not found. (AP0072, AP0073)		void disconnectPort ( in string connectionId) raises (InvalidPort);	
<b>For each instance of the <i>disconnectPort</i> operation found:</b>				
<b>B. Verify that the <i>InvalidPort</i> exception is raised, when the input connectionId parameter is not a known connection to the Port component. (AP0073)</b>				
3. Verify that the <i>InvalidPort</i> exception is raised, when the input connectionId parameter is not a known connection to the Port component.	<b>Pass:</b> The <i>InvalidPort</i> exception is raised when the input connectionId parameter is not a known connection to the Port component. (AP0073) <b>Fail:</b> The <i>InvalidPort</i> exception is not raised when the input connectionId parameter is not a known connection to the Port component. (AP0073)			
<b>C. Verify the presence of an error code value of 2 when the <i>Port</i> name is not found. (C003)</b>				

APP_TC_012				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Verify that the <i>InvalidPort</i> exception error code is 2, when the Port name is not found.	<p><b>Pass:</b> The <i>InvalidPort</i> exception has an error code value of 2 when the Port name is not found. (C003)</p> <p><b>Fail:</b> The <i>InvalidPort</i> exception does not have an error code value of 2 when the Port name is not found. (C003)</p>		<p>Failure of this criterion C003 means failure of the requirements AP0073.</p> <p>The error code of 2 may only be used in the exception being raised from a disconnectPort operation.</p> <p>The error code of 1 may only be used in the exception being raised from a connectPort operation.</p>	
<b>D. Verify that when no exception is raised, the port identified by connectionId is successfully disconnected using the <i>disconnectPort</i> operation. (AP0072)</b>				
5. Verify that the <i>disconnectPort</i> operation uses the connectionId to identify the port to disconnect.	<p><b>Pass:</b> The port identified by the connectionId value is the one being disconnected. (AP0072)</p> <p><b>Fail:</b> The port identified by the connectionId value is not the one being disconnected. (AP0072)</p>			
6. Verify that the identified port is successfully disconnected using this operation.	<p><b>Pass:</b> The identified port is disconnected using this operation. (AP0072)</p> <p><b>Fail:</b> The identified port is not disconnected using this operation. (AP0072)</p>			
<b>End of Test</b>				

Test Recording Log – APP_TC_012			
Step 3 (InvalidPort exception)	Step 4 (errorCode == 2)	Step 5 (connectionId)	Step 6 (Connection broken)



## Test Summary APP\_TC\_012

Once testing is complete for every component of the application 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):**

AP0072 \_\_\_\_\_

AP0073 \_\_\_\_\_

C003 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.13APP\_TC\_013 – PortSupplier::getPort()

### Test Case Number: APP\_TC\_013

PortSupplier::getPort

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0089	The <i>getPort</i> operation shall return the CORBA object reference that is associated with the input port name.

### References

Document Name	Version/Date	Pages referenced
SCA	v2.2.2 05-15-2006	Page 3-12, Section 3.1.3.1.4.5.1.4

### Test Objective

This test case verifies AP0089. The objective of this test is to verify that the *getPort* operation returns the CORBA object reference that is associated with the input port name.

### Places to Verify

Any Resource or component that inherits the Resource interface to obtain a reference to a port.

### IDL References

#### Operation

Object getPort ( in string name )  
raises (CF::PortSupplier::UnknownPort);

### Preconditions

- The source code of the application is available

### Test Description

A. Identify each implementation of the *getPort* operation in the application source code. (AP0089)

1. **Fail:** There are no implementations of the *getPort* operation in the application source code.

For each implementation of the *getPort* operation, perform the following step(s):

- B. Verify that the *getPort* operation returns a CORBA object reference that is associated with the input port name. (AP0089)
1. **Pass:** The *getPort* operation returns a CORBA object reference that is associated with the input port name.
  2. **Fail:** The *getPort* operation does *not* return a CORBA object reference that is associated with the input port name.

## 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.

APP_TC_013				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify each implementation of the <i>getPort</i> operation in the application source code. (AP0089)</b>				
1. Locate the <i>getPort</i> code in the application source code files.	<b>Fail:</b> There are no occurrences of the <i>getPort</i> operation in the application source code. (AP0089)		The SPD file may contain clues about what the name of the source code file is from the name and location of the executable. The <i>getPort</i> source code obtains the uses and provides ports from the SCD and/or SAD files.	
<b>For each implementation of the <i>getPort</i> operation, perform the following step(s):</b>				
<b>B. Verify that the <i>getPort</i> operation returns a CORBA object reference that is associated with the input port name. (AP0089)</b>				
2. Verify in the source code that the <i>getPort</i> operation returns a CORBA Object reference that is associated with the input port name from its parameter.	<b>Pass:</b> The <i>getPort</i> operation returns a CORBA object reference associated with the input port name. (AP0089)  <b>Fail:</b> The <i>getPort</i> operation does not return a CORBA object reference associated with the input port name. AP0089)		The source code might contain a data structure that links each port name with a corresponding CORBA object.	
<b>End of Test</b>				

Test Recording Log- APP_TC_013	
Step 1 List the source code files of <i>getPort</i> operation	Step 2 GetPort returns CORBA object reference? (Pass/Fail?)

## Test Summary APP\_TC\_013

Once testing is complete for every component of the application 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):**

AP0089 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.14APP\_TC\_014 – Port::connectPort raises InvalidPort

### Test Case Number: APP\_TC\_014

Port::connectPort raises InvalidPort

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0070	The <i>connectPort</i> operation shall raise the InvalidPort exception when the input connection parameter is an invalid connection for this port.
C004	The InvalidPort exception indicates one of the following errors has occurred in the specification of a <i>Port</i> association: 1. errorCode 1 means the <i>Port</i> component is invalid (unable to narrow object reference) or illegal object reference,

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-8 to 3-8, Section 3.1.3.1.1.3.1, 3.1.3.1.1.5.1.5

### Test Objective

This test case verifies AP0070 and associated criteria C004. The objective of this procedure is to test the exception requirements of the *connectPort* operation of the Port interface. The test cases also verify that the errorCode value in the raised exception is 1.

### Places to verify

Implementations of the Port interface.

### IDL Reference

#### Exception

```
exception InvalidPort { unsigned short errorCode;  
                        string msg; };
```

#### Operation

```
void connectPort ( in Object connection,  
                  in string connectionId )  
    raises (CF::Port::InvalidPort, CF::Port::OccupiedPort);
```

## Preconditions

- The application source code files are available.

## Test Description

A. Locate all implementations of *connectPort* in the source code. (AP0070)

For each implementation of *connectPort* found in the source code:

- B. Verify that the *connectPort* operation will return an *InvalidPort* exception when the input port reference is invalid (either the input object reference cannot be narrowed or the object reference is illegal). (AP0070)
1. **Fail:** The *connectPort* operation did not throw the *InvalidPort* exception.
  2. **Pass:** The *InvalidPort* exception is thrown. (AP0070)
- C. Verify that the *errorCode* has a value of 1. (C004).
1. **Fail:** The *errorCode* value is not 1.
  2. **Pass:** The *errorCode* value is 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.

APP_TC_014				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all implementations of <i>connectPort</i> in the source code. (AP0070)</b>				
1. Search the application source code for all implementations of <i>connectPort</i> and record the files in which they are found.	<b>Pass:</b> One or more implementations of <i>connectPort</i> are found. (AP0070)  <b>Fail:</b> No implementations of <i>connectPort</i> are found. (AP0070)		Note that we are not interested in any invocations of <i>connectPort</i> .	
<b>For each implementation found, perform the following steps:</b>				
<b>B. Verify that the <i>connectPort</i> operation will return an <i>InvalidPort</i> exception when the input port reference is invalid (either the input object reference cannot be narrowed or the object reference is illegal). (AP0070)</b>				
2. Verify that the <i>connection</i> parameter is validated.	<b>Pass:</b> The <i>connection</i> parameter is validated. (AP0070)		Validation should include verifying that it is a CORBA object and verifying that it is a port object (by narrowing it to a port object)	

APP_TC_014				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify that an <i>InvalidPort</i> exception is thrown if the first parameter fails its validation.	<b>Pass:</b> An <i>InvalidPort</i> exception is thrown when the first parameter is invalid. (AP0070)  <b>Fail:</b> An <i>InvalidPort</i> exception is thrown when the first parameter is valid. (AP0070)  <b>Fail:</b> An <i>InvalidPort</i> exception is not thrown when the first parameter is invalid. (AP0070)			
<b>C. Verify that the <i>errorCode</i> has a value of 1 or 2 (C004).</b>				
4. Verify that the <i>errorCode</i> associated with the <i>InvalidPort</i> exception and raised by the failed <i>connectPort</i> operation has a value of 1.	<b>Pass:</b> the <i>errorCode</i> is set to 1. (C004)  <b>Fail:</b> The <i>errorCode</i> is not set to 1. (C004)		Failure of C004 results in failure of the AP0070 requirement.	
<b>End of Test</b>				

Test Recording Log – APP_TC_014			
Step 1 (implementation files)	Step 2 (first parameter validated)	Step 3 (exception raised)	Step 4 ( <i>errorCode==1</i> )

## Test Summary APP\_TC\_014

Once testing is complete for every component of the application 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):**

AP0070 \_\_\_\_\_

C004 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.15APP\_TC\_015 – Port::connectPort raises OccupiedPort

### Test Case Number: APP\_TC\_015

Port::connectPort

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0069	The <i>connectPort</i> operation shall make a connection to the component identified by its input parameters.
AP0071	The connectPort operation shall raise the OccupiedPort exception when unable to accept the connections because the port is already fully occupied.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-7, 3-8, 3-12, 3-29
SCA AppendixD – Domain Profile	Version 2.2.2 05-15-2006	Pages D-46
SCA AppendixC – Core Framework IDL	Version 2.2.2 05-15-2006	Pages C-13, C-23

### Test Objective

This test verifies AP0069 and AP0071. The objective of this test is to verify that the *connectPort* operation establishes associations between ports and the *OccupiedPort* exception is raised when the identified port is already fully occupied.

### Places to Verify

Applications and any other component that establishes the operations for transferring data and control.

### IDL References

#### Exceptions

exception OccupiedPort{ };

#### Operations

void connectPort (in Object connection, in string connectionId) raises (CF::Port::InvalidPort, CF::Port::OccupiedPort);

## Preconditions

- The application source code files are available.

## Test Description

- A. Locate all implementations of *connectPort* in the source code. (AP0069, AP0071)
  - 1) **Untested:** There are no implementations of *connectPort* in the source code.
- B. Verify that connections are attempted to a valid input port object and that the input *connectionId* is associated with this connection if the connection is successful. (AP0069)
  - 1) **Pass:** The connection is attempted and if successful, the *connectionId* is associated with the connection.
  - 2) **Fail** The connection is not attempted with a valid port object.
  - 3) **Fail:** The connection is successful, but the *connectionId* is not associated with the connection.
- C. Verify that if another connection cannot be accepted because the port is fully occupied, then an *OccupiedPort* exception is raised (AP0071).
  - 1) **Pass:** The port is fully occupied, and the *OccupiedPort* exception is raised.
  - 2) **Fail:** The port is not fully occupied, and the *OccupiedPort* exception is raised.
  - 3) **Fail:** The port is fully occupied, and the *OccupiedPort* exception is not 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 it.

APP_TC_015				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all implementations of <i>connectPort</i> in the source code. (AP0069, AP0071)</b>				
1. Locate all implementations of <i>connectPort</i> in the source code and record the source files containing the implementations.	<b>Untested:</b> No implementations are found. (AP0069, AP0071)			
<b>For each implementation found, perform steps 2 thru 5.</b>				
<b>B. Verify that connections are attempted to a valid input port object, and the input <i>connectionId</i> is associated with this connection if the connection is successful. (AP0069)</b>				
2. Verify that a connection is attempted if the <i>connection</i> parameter is a port.	<b>Pass:</b> A connection is attempted. (AP0069)  <b>Fail:</b> No connection is attempted. (AP0069)			
3. When the connection is successful, verify that the <i>connectionId</i> is associated with this connection.	<b>Pass:</b> The <i>connectionId</i> is associated with the connection. (AP0069)  <b>Fail:</b> The <i>connectionId</i> is not associated with the connection. (AP0069)			
<b>C. Verify that if another connection cannot be accepted because the port is fully occupied, then an <i>OccupiedPort</i> exception is raised (AP0071).</b>				

APP_TC_015				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Verify that, when attempting to connect, there is a test for being fully occupied.	<b>Pass:</b> A fully occupied test is performed. (AP0071) <b>Fail:</b> A fully occupied test is not performed. (AP0071)			
5. Verify that an <i>OccupiedPort</i> exception is raised when a connection cannot be made because the port is fully occupied.	<b>Pass:</b> An <i>OccupiedPort</i> exception is raised when the port is fully occupied. (AP0071) <b>Fail:</b> An <i>OccupiedPort</i> exception is not raised when the port is fully occupied. (AP0071) <b>Fail:</b> An <i>OccupiedPort</i> exception is raised when the port is not fully occupied. (AP0071)			
<b>End of Test</b>				



Test Recording Log – APP_TC_015				
Step 1 (source files)	Step 2 (Connection attempted- Y/N?)	Step 3 ( <i>connectionId</i> associated-Y/N?)	Step 4 (fully occupied test- Y/N?)	Step 5 ( <i>OccupiedPort</i> exception -Y/N?)

## Test Summary APP\_TC\_015

Once testing is complete for every component of the application 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):**

AP0069\_\_\_\_\_

AP0071\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.16APP\_TC\_016 – TestableObject::runTest raises Exceptions

**Test Case Number:** APP\_TC\_016

TestableObject::runTest

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0084	The runTest operation shall not execute any testing when the input testId or any of the input testValues are not known by the component or are out of range
AP0085	The runTest operation shall raise the UnknownTest exception when there is no underlying test implementation that is associated with the input testId given.
AP0086	The runTest operation shall raise the CF UnknownProperties exception when the input parameter testValues contains any CF DataTypes that are not known by the component's test implementation or any values that are out of range for the requested test.
AP0087	The exception parameter <i>invalidProperties</i> shall contain the invalid <i>testValues</i> properties id(s) that are not known by the component or the value(s) are out of range.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-10 to 3-11, Section 3.1.3.1.3

### Test Objective

This test case verifies AP0084, AP0085, AP0086 and AP0087. The objective of this test is to verify that when the application performs the *runTest()* operation invalid inputs are not used and that the proper exceptions are raised. The input parameter testValues is an id/value pair. The CF DataTypes are CORBA IDL structures, which may be used to hold any CORBA basic type or static IDL type. The *id* attribute indicates the kind of value and type (e.g., frequency, preset, etc.). Depending on context, the *id* may be a UUID string, an integer string, or a name identifier. The *value* attribute may be any static IDL type or CORBA basic type.

### Places to Verify

Applications

### IDL References

#### Exceptions

exception UnknownTest{ };

---

exception UnknownProperties {Properties invalidProperties; };

### Operations

void runTest (in unsigned long testId, inout Properties testValues) raises (UnknownTest, UnknownProperties);

### Preconditions

- All application source code files are available.

### Test Description

- A. Locate all instantiations of the *runTest* operation in the source code (AP0084, AP0085, AP0086, AP0087).
  1. **Pass:** The *runTest* code is found.
  2. **Fail:** No *runTest* code is found.
- B. Verify that the *testId* input value is verified, and an *UnknownTest* exception is raised if it is not valid (AP0085).
  1. **Pass:** The *testId* is verified and the *UnknownTest* exception is thrown when the value is not known.
  2. **Fail:** The *testId* is not known, and the *UnknownTest* exception is not thrown.
  3. **Fail:** The *testId* is known, and the *UnknownTest* exception is thrown.
- C. Verify that the *id* field of the input *testValues* is verified and an *UnknownProperties* exception is raised if it is not valid (AP0086).
  1. **Pass:** The *id* field of the input *testValues* is verified and an *UnknownProperties* exception is raised if it is not valid.
  2. **Fail:** The *id* field of the input *testValues* is valid and an *UnknownProperties* exception is raised.
  3. **Fail:** The *id* field of the input *testValues* is invalid and an *UnknownProperties* exception is not raised.
- D. Verify that the *value* field of the input *testValues* is verified and an *UnknownProperties* exception is raised if it is not valid (AP0086).
  1. **Pass:** The *value* field of the input *testValues* is verified and an *UnknownProperties* exception is raised if it is not valid.
  2. **Fail:** The *value* field of the input *testValues* is valid and an *UnknownProperties* exception is raised.
  3. **Fail:** The *value* field of the input *testValues* is invalid and an *UnknownProperties* exception is not raised.
- E. Verify that the *testValues* properties that are not known or out of range are recorded in the *invalidProperties* parameter when the *UnknownProperties* exception is raised. (AP0086, AP0087)
  1. **Pass:** The *runTest* operation raises the *UnknownProperties* exception (AP0086), and the exception keeps track of the property that was erroneously added in the *InvalidProperties* attribute (AP0087).
  2. **Fail:** The *runTest* operation does not raise the *UnknownProperties* exception,
  3. **Fail:** The *InvalidProperties* parameter of the *UnknownProperties* exception was not populated with the erroneously added *id/value* pair.
- F. Verify that no testing is performed when an exception is raised (AP0084).

1. **Pass:** No testing is performed when an exception is raised.
2. **Fail:** Testing is performed when an exception is 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 it.

APP_TC_016				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Locate all instantiations of the <i>runTest</i> operation in the source code (AP0084, AP0085, AP0086, AP0087).</b>				
1. Search the source code for all occurrences of the <i>runTest</i> operation and record the file name.	<b>Pass:</b> <i>runTest</i> source code is found. (AP0084, AP0085, AP0086, AP0087)  <b>Fail:</b> No <i>runTest</i> source code is found. (AP0084, AP0085, AP0086, AP0087)		void CF__Application_impl::runTest ( CORBA::ULong testNum, CF::Properties & testValues)	
<b>B. Verify that the <i>testId</i> input value is verified and an <i>UnknownTest</i> exception is raised if it is not valid (AP0085).</b>				
2. Verify that the <i>testId</i> is validated.	<b>Pass:</b> <i>testId</i> is validated. (AP0085)  <b>Fail:</b> <i>testId</i> is not validated. (AP0085)			
3. Verify that the <i>UnknownTest</i> exception is raised if the <i>testId</i> is not known.	<b>Pass:</b> <i>testId</i> is not known and an <i>UnknownTest</i> exception is raised. (AP0085)  <b>Fail:</b> <i>testId</i> is not known and no exception is raised. (AP0085)			
4. Verify that the <i>UnknownTest</i> exception is not raised if the <i>testId</i> is known.	<b>Pass:</b> <i>testId</i> is known and no exception is raised. (AP0085)  <b>Fail:</b> <i>testId</i> is known and an <i>UnknownTest</i> exception is raised. (AP0085)			
<b>C. Verify that the <i>id</i> field of the input <i>testValues</i> is verified and an <i>UnknownProperties</i> exception is raised if it is not valid (AP0086).</b>				

APP_TC_016				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify that the <i>id</i> field of the <i>testValues</i> is validated.	<b>Pass:</b> <i>id</i> is validated. (AP0086) <b>Fail:</b> <i>id</i> is not validated. (AP0086)		Example: CF::Properties options; options.length(x); options[x].id = PW_SHUTDOWN;	
6. Verify that the <i>UnknownProperties</i> exception is raised if the <i>id</i> field is invalid (i.e. not known).	<b>Pass:</b> <i>id</i> is not known and an <i>UnknownProperties</i> exception is raised. (AP0086) <b>Fail:</b> <i>id</i> is not known and no exception is raised. (AP0086)			
7. Verify that the <i>UnknownProperties</i> exception is not raised if the <i>id</i> field is valid.(i.e. known)	<b>Pass:</b> <i>id</i> is known and no exception is raised. (AP0086) <b>Fail:</b> <i>id</i> is known and an <i>UnknownProperties</i> exception is raised. (AP0086)			
<b>D. Verify that the <i>value</i> field of the input <i>testValues</i> is verified and an <i>UnknownProperties</i> exception is raised if it is not valid (AP0086).</b>				
8. Verify that the <i>value</i> field of the <i>testValues</i> is validated.	<b>Pass:</b> <i>value</i> is validated. (AP0086) <b>Fail:</b> <i>value</i> is not validated. (AP0086)		Example: CF::Properties options; options.length(x);  options[x].value <=<= m_ShutdownCoverage;	
9. Verify that the <i>UnknownProperties</i> exception is raised if the <i>value</i> field is invalid.	<b>Pass:</b> <i>value</i> is not known and an <i>UnknownProperties</i> exception is raised. (AP0086) <b>Fail:</b> <i>value</i> is not known and no exception is raised. (AP0086)			

APP_TC_016				
Steps	Expected Results	Actual Results	Comments	Test Result
10. Verify that the <i>UnknownProperties</i> exception is not raised if the <i>value</i> field is valid.	<b>Pass:</b> <i>value</i> is known and no exception is raised. (AP0086)  <b>Fail:</b> <i>value</i> is known and an <i>UnknownProperties</i> exception is raised. (AP0086)			
<b>E. Verify that the <i>testValues</i> properties that are not known or out of range are recorded in the <i>invalidProperties</i> parameter when the <i>UnknownProperties</i> exception is raised. (AP0086, AP0087)</b>				
11. Verify that when invalid <i>id</i> 's or invalid <i>value</i> 's are detected, they are recorded in the <i>invalidProperties</i> parameter	<b>Pass:</b> <i>id</i> or <i>value</i> is invalid and <i>id</i> and <i>value</i> are recorded in the <i>invalidProperties</i> parameter. (AP0086, AP0087)  <b>Fail:</b> <i>id</i> or <i>value</i> is invalid and <i>id</i> or <i>value</i> is not recorded in the <i>invalidProperties</i> parameter. (AP0086, AP0087)  <b>Fail:</b> <i>id</i> and <i>value</i> are valid and <i>id</i> or <i>value</i> is recorded in the <i>invalidProperties</i> parameter. (AP0086, AP0087)		When an invalid value is detected, both it and its related field should be recorded. This means both an <i>id</i> and a <i>value</i> should be recorded every time.	
<b>F. Verify that no testing is performed when an exception is raised (AP0084).</b>				
12. Verify that no testing is performed before any of the validation found in steps 2 thru 7 occurs.	<b>Pass:</b> No testing is performed before validation. (AP0084)  <b>Fail:</b> Testing is performed before validation. (AP0084)			



APP_TC_016				
Steps	Expected Results	Actual Results	Comments	Test Result
13 If an error is detected, verify that no testing is performed.	<b>Pass:</b> No testing is performed after the detection of an error. (AP0084)  <b>Fail:</b> Testing is performed after the detection of an error. (AP0084)			
End of Test				

Test Recording Log – APP_TC_016							
Step 1 (source code files)	Step 2 ( <i>testId</i> )	Steps 3&4 ( <i>UnknownTest</i> exception raised if <i>testid</i> is unknown-Y/N?)	Step 5 ( <i>id</i> )	Steps 6&7 ( <i>UnknownPr</i> <i>operties</i> exception raised if <i>id</i> is invalid- Y/N?)	Step 8 ( <i>value</i> )	Steps 9&10 ( <i>Unknown</i> <i>Properties</i> exception raised if value is invalid - Y/N?)	Step 13 ( <i>invalidPrope</i> <i>rties</i> )

## Test Summary APP\_TC\_016

Once testing is complete for every component of the application 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):**

AP0084\_\_\_\_\_

AP0085\_\_\_\_\_

AP0086\_\_\_\_\_

AP0087\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.17 APP\_TC\_017 – Resource::Identifier

**Test Case Number:** APP\_TC\_017

Resource::identifier

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0101	The readonly identifier attribute shall contain the unique identifier for a Resource instance.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-16, Section 3.1.3.1.6.4.1
SCA Appendix C	Version 2.2.2 05-15-2006	Page C-11

### Test Objective

This test case verifies AP0101. The objective of this test is to verify that the readonly *identifier* attribute contains the unique identifier for a Resource instance.

### Places to Verify

Resource source code implementation

### IDL References

#### Data

readonly attribute string identifier;

#### Operations

None

### Preconditions

- Domain Profile files are available and have passed the Domain Profile test (APP\_TC\_018).
- The source code files of the applications are available.

## Test Description

A. Search for the implementations of all defined resources. (AP0101)

- a. **Untested:** No Resource implementations can be found.

For each Resource found, perform the following:

B. Verify that the Resource's identifier attribute is in the DCE format, indicating uniqueness. (AP0101)

- a. **Pass:** The Resource's identifier is not in the DCE format.
- b. **Pass:** The Resource's identifier is in the DCE format, indicating uniqueness.

C. Verify that the identifier attribute cannot be modified. (AP0101)

- a. **Fail:** The Resource identifier can be modified.
- b. **Pass:** The Resource identifier is unable to be modified.

## 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 it.

APP_TC_017				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Search for the implementation of all defined resources. (AP0101)</b>				
1. Examine the SAD file and locate the <i>componentinstantiation id</i> declaration. Record the declared ID(s).	<p>The corresponding <i>componentinstantiation id</i> exists and is recorded.</p> <p><b>Untested:</b> The corresponding <i>componentinstantiation id</i> cannot be found. (AP0101)</p>		<p>SAD declaration example:</p> <pre>&lt;partitioning&gt; &lt;componentplacement&gt;   &lt;componentfileref     refid = "xx_File"/&gt;   &lt;componentinstantiation id     = "zzzzz"&gt;</pre>	
<b>For each Resource found perform the following:</b>				
<b>B. Verify that the Resource's identifier attribute is in the DCE format, indicating uniqueness. (AP0101)</b>				
2. Examine the source code and verify that the Resource identifier is set to a value that is in DCE format	<p><b>Pass:</b> The Resource's identifier is in the DCE format, indicating uniqueness. (AP0705)</p> <p><b>Fail:</b> The Resource's identifier is not in the DCE format. (AP0705)</p>		<p>Example:</p> <p>"DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1"</p>	
<b>C. Verify that the identifier attribute cannot be modified. (AP0101)</b>				

APP_TC_017				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify in the source code that the Resource's identifier attribute is only assigned when it is instantiated and not modified or modifiable elsewhere in the code.	<b>Pass:</b> The Resource identifier is unable to be modified. (AP0705) <b>Fail:</b> The Resource identifier can be modified. (AP0705)		<b>NOTE:</b> The readonly quality may be determined in the following ways. <ul style="list-style-type: none"><li>▪ declaring the identifier as static</li><li>▪ In the IDL, use the word readonly in front of the attribute name.</li></ul> The class file that contains the Resource source code should not contain any "setter" functions for the Resource identifier.	
End of Test				

Test Recording Log – APP_TC_017		
Step 1 (Resources)	Step 2 (Resource identifier in DCE format – Y/N?)	Step 3 (Resource identifier able to be modified-Y/N?)



## Test Summary APP\_TC\_017

Once testing is complete for every component of the application 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):**

AP0101\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.18APP\_TC\_018 – Domain Profile

### Test Case Number: APP\_TC\_018

Domain Profile

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0588	Domain Profile files shall be compliant to the <i>Document Type Definitions</i> (DTDs) provided in Appendix D.
AP0590	All XML files shall have as the first two lines as an <i>XML declaration</i> (?xml) and a <i>document type declaration</i> (!DOCTYPE).
AP0591	A <i>Software Package Descriptor</i> file shall have a “.spd.xml” extension.
AP0592	A <i>Software Component Descriptor</i> file shall have a “.scd.xml” extension.
AP0593	A <i>Software Assembly Descriptor</i> file shall have a “.sad.xml” extension.
AP0594	A <i>Properties</i> File shall have a “.prf.xml” extension.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	v2.2.2 05-15-2006	Pages 3-90thru 3-92
SCA AppendixD Domain Profile	v2.2.2 05-15-2006	Pages D1-D13(SPD), D19-D28(PRF), D28-D32(SCD), D33-D47(SAD)
Attachment I to AppendixD DomainProfile Document Type Definitions	Version 2.2.2 05-15-2006	

### Test Objective

This test verifies AP0588, AP0590, AP0591, AP0592, AP0593, and AP0594. The objective of this test is to verify that the Domain Profile is compliant to the Document Type Definitions (DTDs) provided in Appendix D. The test also verifies that the Domain Profile files have the correct file extensions and that the first two lines of each Domain Profile file has the proper declaration.

If verifying SCA Extension requirements test case APP\_TC\_051 replaces this test case.

### Places to Verify

Domain Profile files.

## IDL References

None.

## Preconditions

- All the Domain profile files are available
- The SCA v2.2.2 DTD files are available to use for verification.
  - softwareassembly.2.2.2.dtd
  - softpkg.2.2.2.dtd
  - softwarecomponent.2.2.2.dtd
  - properties.2.2.2.dtd

## Test Description

The following XML files, file name extensions, and DTD files will be verified.

Domain Profile Types and Applicable SCA DTDs		
Doman Profile File Name	File Name Extension	DTD File
Software Assembly Descriptor (SAD)	sad.xml	softwareassembly.2.2.2.dtd
Software Package Descriptor (SPD)	spd.xml	softpkg.2.2.2.dtd
Software Component Descriptor (SCD)	scd.xml	softwarecomponent.2.2.2.dtd
Properties Descriptor (PRF)	prf.xml	properties.2.2.2.dtd

- A. Identify the Domain Profile files and their type used in the application under test and verify that there is a corresponding DTD file using the above table (AP0588).
  1. **Fail:** Not all Domain Profile file type have a corresponding DTD.
  2. **Pass:** Each Domain Profile file type has a corresponding DTD.
- B. Verify that the *Domain Profile* files are compliant to the applicable SCA DTD (AP0588).
  1. **Fail:** The *Domain Profile* files are not compliant to the DTD files specified in the SCA.
  2. **Pass:** The *Domain Profile* files are compliant to the DTD files specified in the SCA.
- C. Verify that each DTD file is installed in the domain and has ".dtd" as its filename extension (AP0589).
  1. **Fail:** The DTD file is not installed in the domain or does not have ".dtd" as its filename extension.
  2. **Pass:** The DTD file is installed in the domain and has ".dtd" as its filename extension

- D. Verify that the *Domain Profile* file has the first two lines with *XML* declaration (`?xml`) and a *document type declaration* (`!DOCTYPE`) (AP0590).
1. **Fail:** The *Domain Profile* file does not have the *XML or DTD* type declaration.
  2. **Pass:** The *Domain Profile* file has the *XML and DTD* type declarations.
- E. Verify that each Domain Profile file has the correct file name extension as indicated in the table above (AP0591, AP0592, AP0593, AP0594).
1. **Fail:** Not all Domain Profile files have the correct file name extension.
  2. **Pass:** Each Domain Profile file has the correct file name extension.

## 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 it.

APP_TC_018				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the Domain Profile files and their type used in the application under test and verify that there is a corresponding DTD file using the above table (AP0588).</b>				
1. Obtain and record the directory where the Domain Profile (XML) files are located.	<b>Untested:</b> The directory is not located. <b>Pass:</b> The directory is located.			
2. Identify the type of each Domain Profile (XML) file used in the application. The types are SAD, SPD, SCD, and PRF.	The following types should exist. xxx.sad.xml -> file is required (SAD) xxx.spd.xml -> file(s) are required (SPD) xxx.scd.xml -> file(s) are optional (SCD) xxx.prf.xml -> file(s) are optional (PRF)  <b>Pass:</b> SAD and SPD files found. (AP0588)  <b>Fail:</b> No SAD or SPD files found. (AP0588)		This can be done by examining the XML file extensions as indicated in the expected results column.	
3. Locate the corresponding DTD file from the application directory for each Domain Profile file type identified in Step 2.	<b>Pass:</b> Each type of Domain Profile file has a corresponding DTD file. (AP0588)  <b>Fail:</b> Not all Domain Profile files have a corresponding DTD file. (AP0588)		Mapping example: <u>XML file types</u> <u>Matching DTD Files</u> xxx.sad.xml        softwareassembly.2.2.2.dtd xxx.spd.xml        softpkg.2.2.2.dtd xxx.scd.xml        softwarecomponent.2.2.2.dtd xxx.prf.xml        properties.2.2.2.dtd	

APP_TC_018				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>B. Verify that the <i>Domain Profile</i> files are compliant to the applicable SCA DTD. (AP0588).</b>				
4. Compare each DTD file from Step 3 to the applicable SCA DTD file in attachment 1 for compliance.	<p><b>Pass:</b> There are only white space differences between the files. (AP0588)</p> <p><b>Fail:</b> There is more than one white space difference between the files. (AP0588)</p>		<p>White space differences include spaces, tabs, line feeds, and carriage returns.</p> <p>A file comparison tool such as the diff function in UNIX/Linux or cdiff with windows can be used for this test.</p>	
<b>C. Verify that each DTD file is installed in the domain and has ".dtd" as its filename extension (AP0589).</b>				
5. Examine each installed DTD file from Step 3 and verify that the filename extension is ".dtd".	<p><b>Pass:</b> All DTD files have the file extension ".dtd". (AP0589)</p> <p><b>Fail:</b> One or more DTD files do not have the file extension ".dtd". (AP0589)</p>			
<b>D. Verify that the <i>Domain Profile</i> file has the first two lines with XML declaration (?xml) and a document type declaration (!DOCTYPE) (AP0590).</b>				
6. Examine each Domain Profile (XML) file and verify that the first two lines are XML declaration and !DOCTYPE declaration.	<p><b>Pass:</b> The first two lines are XML and !DOCTYPE declarations. (AP0590)</p> <p><b>Fail:</b> The first two lines are not XML or !DOCTYPE declarations. (AP0590)</p>		<p>Sample declaration:</p> <pre>&lt;?xml version="1.0" ...&gt; &lt;!DOCTYPE ...&gt;</pre>	
<b>E. Verify that each Domain Profile file has the correct file name extension (AP0591, AP0592, AP0593, AP0594).</b>				

APP_TC_018				
Steps	Expected Results	Actual Results	Comments	Test Result
7. Verify each <i>Software Package Descriptor</i> (SPD) file has a “.spd.xml” extension.	<p><b>Pass:</b> Each <i>Software Package Descriptor</i> (SPD) file has a “.spd.xml” extension. (AP0591)</p> <p><b>Fail:</b> One or more <i>Software Package Descriptor</i> (SPD) files does not have a “.spd.xml” extension. (AP0591)</p>			
8. Verify each <i>Software Component Descriptor</i> (SCD) file has a “.scd.xml” extension.	<p><b>Pass:</b> Each <i>Software Component Descriptor</i> (SCD) file has a “.scd.xml” extension. (AP0592)</p> <p><b>Fail:</b> One or more <i>Software Component Descriptor</i> (SCD) files does not have a “.scd.xml” extension. (AP0592)</p>			
9. Verify each <i>Software Assembly Descriptor</i> (SAD) file has a “.sad.xml” extension.	<p><b>Pass:</b> Each <i>Software Assembly Descriptor</i> (SAD) file has a “.sad.xml” extension. (AP0593)</p> <p><b>Fail:</b> One or more <i>Software Assembly Descriptor</i> (SAD) files does not have a “.sad.xml” extension. (AP0593)</p>			
10. Verify each <i>Properties Descriptor</i> (PRF) file has a “.prf.xml” extension.	<p><b>Pass:</b> Each <i>Properties Descriptor</i> (PRF) file has a “.prf.xml” extension. (AP0594)</p> <p><b>Fail:</b> One or more <i>Properties Descriptor</i> (PRF) files does not have a “.prf.xml” extension. (AP0594)</p>			
<b>End of Test</b>				

Test Recording Log – APP_TC_018					
Step 1 (directory containing the Domain Profile files)					
Step 2 (Domain Profile file type)	Step 3 (DTD file)	Step 4 (Compliant to SCA DTD)	Step 5 (all DTD files have dtd extension)	Step 6 (?XML and !DOCTYPE declarations)	Steps 7 thru 10 (proper file extension)
SAD Y/N	SAD Y/N	SAD Y/N	SAD Y/N	SAD Y/N	“.sad.xml” extension Y/N
SPD Y/N	SPD Y/N	SPD Y/N	SPD Y/N	SPD Y/N	“.spd.xml” extension Y/N
SCD Y/N	SCD Y/N	SCD Y/N	SCD Y/N	SCD Y/N	“.scd.xml” extension Y/N
PRF Y/N	PRF Y/N	PRF Y/N	PRF Y/N	PRF Y/N	“.prf.xml” extension Y/N



## Test Summary APP\_TC\_018

Once testing is complete for every component of the application 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):

AP0588\_\_\_\_\_

AP0590\_\_\_\_\_

AP0591\_\_\_\_\_

AP0592\_\_\_\_\_

AP0593\_\_\_\_\_

AP0594\_\_\_\_\_

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

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

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

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

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

## B.19APP\_TC\_019 – LifeCycle::releaseObject()

**Test Case Number:** APP\_TC\_019

LifeCycle::releaseObject

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0075	The releaseObject operation shall release all internal memory allocated by the component during the life of the component.
AP0076	The releaseObject operation shall tear down the component and release it from the CORBA environment.
AP0078	The releaseObject operation shall raise a ReleaseError exception when a release error occurs.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-9, Section 3.1.3.1.2.3.2 Page 3-10, Section 3.1.3.1.2.5.2
SCA Appendix C - IDL	Version 2.2.2 05-15-2006	Page C-14
Minimum CORBA Specification	Version 1.0 02-08-2001	Pages 1-12; 1-13; 1-26; 1-27

### Test Objective

This test case verifies AP0075, AP0076 and AP0078. The objective of this test is to verify that the *releaseObject()* operation releases all the internal memory allocated by it, tears down the component from the COBRA environment, and raises an exception when an error occurs.

### Places to Verify

All Resources with the *releaseObject* operation.

### IDL References

#### Exception

```
exception ReleaseError { CF::StringSequence errorMessages; };
```

#### Operation

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

## Preconditions

- The source code files for the application are available.

## Test Description

Use the application name to acquire the running application and obtain references to all of its components.

For each Resource:

- A. Verify that each component has a corresponding memory deallocation operation for each allocation of memory. (AP0075)
  1. **Pass:** For each allocation of memory, there exists source code to deallocate the memory.
  2. **Fail:** For one or more instances where memory is allocated, there does not exist source code to deallocate the memory.
- B. Verify that the *releaseObject* operation tears down and releases the component from the CORBA environment. (AP0076)
  1. **Untested:** Unable to obtain the object reference.
  2. **Pass:** The *releaseObject* operation tears down the component and releases it from the CORBA environment.
  3. **Fail:** The *releaseObject* operation does not tear down the component or release it from the CORBA environment.
- C. Verify that the *releaseObject* operation raises an exception when a release error occurs. (AP0078)
  1. **Pass:** The *releaseObject* operation raises a *ReleaseError* exception.
  2. **Fail:** The *releaseObject* operation does not raise a *ReleaseError* exception.

## Manual Test Steps

Notes: 1. Test Results can be P, F, U, N/A (Pass, Fail, Untested, Not Applicable, or any other as appropriate).  
 2. The Test Recording Log is intended to record long comments and lists of SPD files, SCD files, PRF, and other file/data.

APP_TC_019				
Steps	Expected Results	Actual Results	Comments	Test Result
Use the application name to acquire the running application and obtain references to all of its components. For each Resource:				
A. Verify that each component has a corresponding memory <i>deallocation</i> operation for each allocation of memory. (AP0075)				
1. Perform a keyword search for <i>releaseObject</i> operation on all source code provided by the developer. Identify any references to the other source code file(s). Record the source code file(s).	The <i>releaseObject</i> operation exists in the source code file.		Example:  <pre>void CF__Application_impl::   releaseObject   (     CORBA::Environment&amp;     _env   ) {   INSTRUMENTATION_UTILITIES_ENTRY_EXIT("CF__Application_impl",     "releaseObject",     "CORBA::Environment&amp;"); }</pre>	
2. Ensure calling <i>releaseObject</i> results in the application component being torn down.	<p><b>Pass:</b> Every allocation has a corresponding deallocation and every deallocation has a corresponding allocation. (AP0075)</p> <p><b>Fail:</b> There is at least one allocation that does not have a corresponding deallocation. (AP0075)</p>		Looking for matching memory operations allocation/ <i>deallocation</i> . Non-matching pairs will FAIL this requirement.	
B. Verify that the <i>releaseObject</i> operation tears down and releases the component from the CORBA environment. (AP0076)				

APP_TC_019				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Locate the object reference to the resource which will be released.	<b>Untested:</b> Unable to obtain the object reference. (AP0076)		Examples to search for:  'release', 'destroy', 'deactivate_object',  Destroy the naming context and unbinding it from its parent naming context.  m_varNamingContext-> <b>destroy</b> ( _env);	
4. Examine the source code files and verify that the <i>releaseObject</i> operation tears down and releases an instantiated component from the CORBA environment.	<b>Pass:</b> The <i>releaseObject</i> operation tears down the component and releases it from the CORBA environment. (AP0076)  <b>Fail:</b> The <i>releaseObject</i> operation does not tear down the component or release it from the CORBA environment. (AP0076)		Examples:  <b>releaseResources</b> ( errorMsg, _env );  <b>releaseComponents</b> ( errorMsg, _env );	
<b>C. Verify that the <i>releaseObject</i> operation raises an exception when a release error occurs. (AP0078)</b>				
5. Examine the source code files and verify that the <i>ReleaseError</i> exception is raised when a release error occurs.	<b>Pass:</b> The <i>releaseObject</i> operation raises a <i>ReleaseError</i> exception. (AP0078)  <b>Fail:</b> The <i>releaseObject</i> operation does not raise a <i>ReleaseError</i> exception. (AP0078)		Example:  CF::LifeCycle::ReleaseError_ptr pException = new CF::LifeCycle::ReleaseError( ErrorMessage );  THROW ( pException );	
<b>End of Test</b>				

Test Recording Log – APP_TC_019				
Step 1 (source code file)	Step 2 (Items having matching allocation/deallocation operations)	Step 4 (release Object tears down component-Y/N?)	Step 5 (Release Error exception raised- Y/N?)	

## Test Summary APP\_TC\_019

Once testing is complete for every component of the application 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):**

AP0075\_\_\_\_\_

AP0076\_\_\_\_\_

AP0078\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.20APP\_TC\_020 – AEP, Ada Functionality

**Test Case Number:** APP\_TC\_020

Application::Ada Functionality

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0655	Any Ada application shall be restricted to using the equivalent Ada functionality, as defined in POSIX Ada language binding (ISO/IEC 14519:2001), designated as mandatory by the AEP or may use the C interface.

### References

Document Name	Version/Date	Location (Pages, Section)
Information Technology – POSIX Ada Language Interfaces – Binding for System Application Program Interface (API), ISO/IEC 14519, IEEE Std 1003.5	Second Edition 12-15-2001.	
SCA Appendix B (SCA Application Environment Profile (AEP))	Version 2.2.2 05-15-2006	Pages B-1 – B-18

### Test Objective

This test case verifies AP0655. The objective of this test is to verify that if the application is using the Ada language then it is restricted to using equivalent Ada functionality as defined in the ISO/IEC 14519:2001 POSIX Ada Language Interface specification. This can be done by implementing a C/C++ interface as described in Appendix B.

### Places to Verify

Applications having POSIX Ada language

### IDL References

None.

### Preconditions

- The application source code files are available.



## Test Description

For each component:

A. Verify that if Ada is implemented, that the interface is restricted to using the equivalent Ada functionalities as referenced in the POSIX Ada Language Binding specification (AP0655).

1. **N/A:** Application is not using Ada functionalities.
2. **Pass:** Application is restricted to using Ada functionalities as referenced in the POSIX Ada Language Binding specification.
3. **Fail:** Application is not restricted to using Ada functionalities as referenced in the POSIX Ada Language Binding specification.

## 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 it.

APP_TC_020				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Verify that if Ada is implemented, that the interface is restricted to using the equivalent Ada functionalities as referenced in the POSIX Ada Language Binding specification (AP0655).</b>				
1. Search for files that end in “ada”, “ads” and “adb” (the case does not matter). Record the source file(s) name(s).	<b>N/A: Application is not using Ada functionalities. (AP0655)</b>		Ask the developer for Ada source file extensions.	
2. Locate and examine the POSIX interface to Ada. If using C language, skip and go to the next source file. If POSIX Ada Language Interface is used, ensure compliance as stated in SCA 222 Appendix B.	<b>Pass:</b> Application is restricted to using Ada functionalities as referenced in the specification. (AP0655)  <b>Fail:</b> Application is not restricted to using Ada functionalities as referenced in the specification. (AP0655)		Review the source code against the AEP list of Appendix B.  Items in the tables are labeled either MAN or RNQ. MAN means the OE needs to supply them (so they would not be in an application). NRQ means they are not required and an application cannot use it.	
<b>End of Test</b>				

Test Recording Log – APP_TC_020	
Step 1 (Ada source file name)	Step 2 (Interface restricted to Ada functionalities – Y/N?)

## Test Summary APP\_TC\_020

Once testing is complete for every component of the application 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):**

AP0655 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.21APP\_TC\_021 – High Order Languages

**Test Case Number:** APP\_TC\_021

General Software Rules::New Software

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0628	Software developed for an SCA-compliant system shall be developed in a standard higher order language.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	v2.2.2 05-15-2006	Page 3-98, Section 3.4.1.1

### Test Objective

This test case verifies AP0628. The objective of this test is to verify that waveform applications are written in a standard higher order language.

The definition to differentiate a standard higher (vice a ‘high’) order language vice a lower order language is inherently relative. Standard higher order language can be described as a programming language that has a higher level of abstraction and is substantially easier to code (than machine language) (although not necessarily able to run more efficiently and quicker). A higher order language does not require the programmer to directly handle registers, memory addresses or opcodes. It is preferable for this test case to provide the guidance of what a ‘standard higher order language’ is rather than providing an explicit list.

In summary, for our purposes in this test case, a higher order language is:

- A general-purpose programming language that allows programs to be written without having to understand the inner workings of a computer.
- Translated, using a computer program called a “compiler”, into a format (object code) executable by a computer.

### Places to Verify

Waveform applications.

### IDL References

None.

## Preconditions

- The application source code files are available.

## Test Description

For each application source code file:

- A. Verify manually that the non-legacy code has been developed with a standard higher order language as described in the Test Objective section (AP0628).
  1. **Pass:** Code has been developed with a standard higher order programming language.
  2. **Fail:** Code has not been developed with a standard higher order programming language.

## Manual Test Steps

- Notes: 1. It is highly recommended that the waveform application software developers are involved in the testing.  
 2. Test Result will include Pass, Fail, Untested, or N/A.  
 3. The Test Recording Log is intended to record data for each step that requires it.

APP_TC_021				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Verify manually that the non-legacy code has been developed with a standard higher order language as described in the Test Objective section (AP0628).</b>				
1. Identify and record which application code is 'non-legacy' source code. Note: Have the developer provide documentation as to what is legacy software.	<b>Identified</b> 'non-legacy' source code, which must comply with this requirement. (AP0628)		Note: 'legacy' source code is defined per JTNC Standards as "software components of the radio that were produced before the software code development for SCA compliance was started."	
2. Verify that the application source code within the SCA boundary was developed in a standard higher order language.	<b>Pass:</b> The application source code within the SCA boundary was developed in a standard higher order language. (AP0628)  <b>Fail:</b> The application source code within the SCA boundary was not developed in a standard higher order language. (AP0628)		Look for file suffixes such as dot-dat or dot-asm, and other suffixes not recognized. They may indicate a non-higher order language. Consult with the developer as to what non-higher order language code they are using.  Examples:  Assembly and VHDL languages	
<b>End of Test</b>				

Test Recording Log – APP_TC_021	
Step 1 (non-legacy code, such as name of files, function, procedures, etc.)	Step 2 (What Higher Order Language was used?)



## Test Summary APP\_TC\_021

Once testing is complete for every component of the application 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):**

AP0628\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.22APP\_TC\_022 – AEP, Abnormal Termination

**Test Case Number:** APP\_TC\_022

AEP abnormal termination

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0661	An application that conforms to the AEP shall not result in abnormal termination of the process because this profile does not support multiple processes.

### References

Document Name	Version/Date	Location (Pages/Section)
SCA	Version 2.2.2 05-15-2006	Page 3-95, Section 3.2.1.1, paragraph 1
SCA Appendix B, Application Environment Profile (AEP)	Version 2.2.2 05-15-2006	Page B-4 to B-5; Section B.4.1.4
IEEE Standard for Information Technology – Standardized Application Environment Profile (AEP) – POSIX® Realtime and Embedded Application Support, IEEE Std 1003.13-2003.	09-10-2004	Pages 43, Section 6.3.1

### Test Objective

This test case verifies AP0661. Table B-6 of the SCA Appendix B, AEP specifies the mandatory POSIX Signals function to be provided by the OE. This test case should only be executed if the application waveform provides any of the signals functions in Table B-6 in which the functions over-ride the base POSIX functions provided by the OE. The SCA AEP specification requires that these signals functions be handled properly so other processes are not impacted. The objective of this test is to verify that the application waveform, which is the provider of these signals' functions, implements signal handlers for each signal function, in order to properly manage process terminations in an environment that does not support multiple processes.

### Places to Verify

Application

### IDL References

None

## Preconditions

- The application waveform is the provider of one or more signals functions listed in Table B-6 of the SCA Appendix B, AEP.
- Complete set of header files for the application environment are available.

## Test Description

A. Identify the source code and/or header files that implement POSIX Signal functions listed in Table B-6. (AP0661)

1. **Pass:** The source code and/or header files are available.
2. **Untested:** The source code files/ or header files are not available. If untested, then stop test.

For each POSIX Signal functions identified in Step A, perform the following:

B. Verify there is a signal handler for each of the POSIX signal functions. (AP0661)

1. **Pass:** All signal functions has a corresponding signal handler.
2. **Fail:** One or more signal function(s) do not have a corresponding signal handler.

## 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 it.

APP_TC_022				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify the source code and/or OE Header files that implement POSIX Signal functions in Table B-6. (AP0661)</b>				
1. Perform a search for header files for each signal function. Refer to list of functions in the Comments cell.	<p><b>Pass:</b> The source code and/or header files are available for all functions. (AP0661)</p> <p><b>Untested:</b> The source code files are not available for one or more function(s). (AP0661)</p>		abort() kill() pause() raise() sigaction() sigaddset() sigdelset() sigemptyset() sigfillset() sigismember() signal() sigpending() sigprocmask() sigsuspend() sigwait()	
<b>For each POSIX Signal functions identified in Step A, perform the following:</b>				
<b>B. Verify there is a signal handler for the POSIX signal functions. (AP0661)</b>				

APP_TC_022				
Steps	Expected Results	Actual Results	Comments	Test Result
2. For each signal function, verify that is a signal handler to ensure that only the thread specified for execution of the signal function is executed.	<b>Pass:</b> Signal handlers are implemented for each POSIX signal. (AP0661)  <b>Fail:</b> One or more signal(s) do not have a signal handler. (AP0661)		The provider of signals functions is expected to safeguard the application waveform from any abnormal termination of processes when a signal function is called on a single process. Signal handlers are generally implemented to intercept the behavior of the signal (which is to interrupt a process' normal flow). If a process registers to a signal handler, then its routine is executed instead.	
End of Test				

Test Recording Log – APP_TC_022	
Step 1 (source file having POSIX signal functions)	Step 2 (List signal handler for each signal function.

## Test Summary APP\_TC\_022

Once testing is complete for every component of the application 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):**

AP0661 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.23APP\_TC\_024 – Minimum CORBA

**Test Case Number:** APP\_TC\_024

CORBA and CORBA Services::Applications

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0607	Applications shall be limited to using CORBA and CORBA services defined in the referenced minimumCORBA specification[5?]

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	v2.2.2 05-15-2006	Page 3-95, Section 3.2.1.2
Minimum CORBA Specification	V 1.0 08-01-2002	All
Cygwin Unix-like software environment	v 1.7.x or later	<a href="http://www.cygwin.com/">http://www.cygwin.com/</a>
AP0607 Procedure.xls		Obtainable via JTEL test engineer POC

### Test Objective

This test case verifies AP0607. The objective of this test is to verify that the application source code is limited to using CORBA as referenced in the Minimum CORBA specification of the SCA v2.2.2. There are two test methods available to test AP0607. One test method should be executed to perform this test. Choose the test method based on resource availability.

Test Method 1 employs a grep search tool developed by JTEL that performs keyword search for the disallowed CORBA calls per the Minimum CORBA Specification. Using Test Method 1 requires the following tool/files: 1) Cygwin, 2) Search Spreadsheet (AP0607 Procedure.xls).

Test Method 2 employs any independent keyword search tool for searching the disallowed CORBA calls per the Minimum CORBA Specification. Using Test Method 2 requires the following tool/files: 1) Any keyword search tool, 2) Search Spreadsheet (AP0607 Procedure.xls).

### Places to Verify

Waveform application source code for all of the components.

### IDL References

None



## Preconditions

- The source code files of the product undergoing test are available.
- Cygwin simulated UNIX environment is installed on the testing machine. [Test Method 1].
- The latest version of “AP0607 Procedure.xls” spreadsheet is available. [Test Method 1 or Test Method 2]

## Test Description

### Test Method 1: JTEL Search Script

- A. Prepare the test environment for the Search Script.
- B. Create the script file.
- C. Execute the Search Script.
- D. Review the results of the Search Script for minimumCORBA violation. (AP0607)
  1. **Pass:** Violation of the minimumCORBA Specification is not found.
  2. **Fail:** Violation of the minimumCORBA Specification is found.

### Test Method 2: Independent Keyword Search Tool

- A. Prepare the test environment for manual search and compare CORBA calls.
- B. Execute keyword search using provided CORBA call list.
- C. Review the search results for minimumCORBA violation. (AP0607)
  1. **Pass:** Violation of the minimumCORBA Specification is not found.
  2. **Fail:** Violation of the minimumCORBA Specification is found.

APP\_TC\_024 Table 1

## List of violations of Minimum CORBA

_this	discard_requests	InterfaceDefSeq	RequestProcessingPolicy_ptr
AdapterActivator	discriminator_type	InterfaceDefSeq_ptr	RequestProcessingPolicy_var
AdapterActivator_ptr	DynAny	InterfaceDefSeq_var	RequestProcessingPolicyValue
AdapterActivator_var	DynAny_ptr	IObject	RequestProcessingPolicyValue_ptr
AdapterInactive	DynAny_var	IObject_ptr	RequestProcessingPolicyValue_var
AliasDef	DynArray	IObject_var	SecConstruction
AliasDef_ptr	DynArray_ptr	is_a	SequenceDef
AliasDef_var	DynArray_var	is_a	SequenceDef_ptr
AnySeq	DynEnum	length	SequenceDef_var
AnySeq_ptr	DynEnum_ptr	member_count	ServantActivator
AnySeq_var	DynEnum_var	member_label	ServantActivator_ptr
ArrayDef	DynFixed	member_name	ServantActivator_var
ArrayDef_ptr	DynFixed_ptr	member_type	ServantLocator
ArrayDef_var	DynFixed_var	members	ServantLocator_ptr
AttrDescriptionSeq	DynSequence	MessageInterceptor	ServantLocator_var
AttrDescriptionSeq_ptr	DynSequence_ptr	MessageInterceptor_ptr	ServantManager
AttrDescriptionSeq_var	DynSequence_var	MessageInterceptor_var	ServantManager_ptr
AttributeDef	DynStruct	ModuleDef	ServantManager-var
AttributeDef_ptr	DynStruct_ptr	ModuleDef_ptr	ServantRetentionPolicy
AttributeDef_var	DynStruct_var	ModuleDef_var	ServantRetentionPolicy_ptr
AttributeDescription	DynUnion	ModuleDescription	ServantRetentionPolicy_var
AttributeDescription_ptr	DynUnion_ptr	ModuleDescription_ptr	ServantRetentionPolicyValue
AttributeDescription_var	DynUnion_var	ModuleDescription_var	ServantRetentionPolicyValue_ptr
AttributeMode	EnumDef	NameValuePair	ServantRetentionPolicyValue_var
AttributeMode_ptr	EnumDef_ptr	NameValuePair_ptr	set_servant
AttributeMode_var	EnumDef_var	NameValuePair_var	set_servant_manager
Bounds	EnumMemberSeq	NameValuePairSeq	shutdown
ConstantDef	EnumMemberSeq_ptr	NameValuePairSeq_ptr	StringDef
ConstantDef_ptr	EnumMemberSeq_var	NameValuePairSeq_var	StringDef_ptr
ConstantDef_var	equal	non_existent	StringDef_var
ConstantDescription	ExcDescriptionSeq	NoServant	StructDef
ConstantDescription_ptr	ExcDescriptionSeq_ptr	OpDescriptionSeq	StructDef_ptr
ConstantDescription_var	ExcDescriptionSeq_var	OpDescriptionSeq_ptr	StructDef_var

APP\_TC\_024 Table 1

## List of violations of Minimum CORBA

ConstructionPolicy	ExceptionDef	OpDescriptionSeq_var	StructMember
ConstructionPolicy_ptr	ExceptionDef_ptr	OperationDef	StructMember_ptr
ConstructionPolicy_var	ExceptionDef_var	OperationDef_ptr	StructMember_var
Contained	ExceptionDefSeq	OperationDef_var	StructMemberSeq
Contained_ptr	ExceptionDefSeq_ptr	OperationDescription	StructMemberSeq_ptr
Contained_var	ExceptionDefSeq_var	OperationDescription_ptr	StructMemberSeq_var
ContainedSeq	ExceptionDescription	OperationDescription_var	the_activator
ContainedSeq_ptr	ExceptionDescription_ptr	OperationMode	ThreadPolicy
ContainedSeq_var	ExceptionDescription_var	OperationMode_ptr	ThreadPolicy_ptr
Container	FieldName	OperationMode_var	ThreadPolicy_var
Container_ptr	fixed_digits	ORB	ThreadPolicyValue
Container_var	fixed_scale	ORB_ptr	TypedefDef
Container_ptr	FixedDef	ORB_var	TypedefDef_ptr
content_type	FixedDef_ptr	param_count	TypedefDef_var
ContextIdSeq	FixedDef_var	parameter	TypeDescription
ContextIdSeq_ptr	ForwardRequest	ParameterDescription	TypeDescription_ptr
ContextIdSeq_var	get_current	ParameterDescription_ptr	TypeDescription_var
create_alias_tc	get_default_context	ParameterDescription_var	UnionDef
create_array_tc	get_implementation	ParameterMode	UnionDef_ptr
create_enum_tc	get_interface	ParameterMode_ptr	UnionDef_var
create_exception_tc	get_servant_manager	ParameterMode_var	UnionMemberSeq
create_fixed_tc	get_service_information	ParDescriptionSeq	UnionMemberSeq_ptr
create_implicit_activation_policy	hold_requests	ParDescriptionSeq_ptr	UnionMemberSeq_var
create_interface_tc	IdentifierContextIdentifier	ParDescriptionSeq_var	VersionSpec
create_list	IdentifierContextIdentifier_ptr	perform_work	VersionSpec_ptr
create_operation_list	IdentifierContextIdentifier_var	PrimitiveDef	VersionSpec_var
create_recursive_sequence_tc	IDLType	PrimitiveDef_ptr	work_pending
create_request	IDLType_ptr	PrimitiveDef_var	WstringDef
create_request_processing_policy	IDLType_var	PrimitiveKind	WstringDef_ptr
create_sequence_tc	ImplicitActivationPolicy	PrimitiveKind_ptr	WstringDef_var
create_servant_retention_policy	ImplicitActivationPolicy_ptr	PrimitiveKind_var	
create_string_tc	ImplicitActivationPolicy_var	Repository	
create_struct_tc	ImplicitActivationPolicyValue	Repository_ptr	

APP_TC_024 Table 1			
List of violations of Minimum CORBA			
create_thread_policy	ImplicitActivationPolicyValue_ptr	Repository_var	
create_union_tc	ImplicitActivationPolicyValue_var	RepositoryIdSeq	
create_wstring_tc	Interceptor	RepositoryIdSeq_ptr	
deactivate	Interceptor_ptr	RepositoryIdSeq_var	
default_index	Interceptor_var	RequestInterceptor	
DefinitionKind	InterfaceDef	RequestInterceptor_ptr	
DefinitionKind_ptr	InterfaceDef_ptr	RequestInterceptor_var	
DefinitionKind_var	InterfaceDef_var	RequestProcessingPolicy	

## 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 it.

APP_TC_024				
Steps	Expected Results	Actual Results	Comments	Test Result
Test Method 1				
A. Prepare the test environment for the Search Script.				
1. In the "AP0607 Procedure" spreadsheet: Clear all columns from the "Working" tab. This is a working space to prepare for the search script to run.	The "Working" sheet should be blank.			
2. In the "AP0607 Procedure" spreadsheet: Select all of Column A from the "minCORBA Violations" tab. Copy that column and paste it into Column B of the "Working" tab. Go to the bottom of the copy and perform another paste into column B. This is so every "violation" appears twice in the column.	Column B contains the full list of violation calls.			
3. In the "AP0607 Procedure" spreadsheet: Perform an ascending Sort of the "Working" tab. Now all items are alphabetized and each item appears twice in the column. This is your Minimum CORBA violation list.	Column B sorted by alphabetized and each violation call appears twice.			

APP_TC_024				
Steps	Expected Results	Actual Results	Comments	Test Result
4. In the “AP0607 Procedure” spreadsheet: Select the entire column B of the “Working” tab. Copy that column and paste it into column C of the “minCORBA Search Script” tab.	Column C contains the same list as Column B.			
5. In the “AP0607 Procedure” spreadsheet: Finally, check that the new datum corresponds to the commands in the other columns of the “minCORBA Search Script” tab. Add (or subtract) command parameters in columns A, B, D, E, F and G until each entry in column C has the proper command parameters.				
<b>B. Create the script file</b>				
6. In the “AP0607 Procedure” spreadsheet: Click on the “minCORBA Search Script” tab.				
7. In the “AP0607 Procedure” spreadsheet: Select and Copy all rows of datum from columns A to G of the worksheet.				
8. Open a blank(empty) text file in Notepad and paste into the file the information from columns A to G.	The contents of “minCORBA” tab are copied onto a text file.			

APP_TC_024				
Steps	Expected Results	Actual Results	Comments	Test Result
9. Remove “tab” characters from the file using the ‘find & replace’ capability. a) Menu select Edit -> Replace. A dialog box for this appears. b) Find a tab character in your file. Select it, copy it and then paste it into the “Find What” part of the dialog box c) Click the “Replace All” button.	All tab characters are removed.			
10. Save the text file. It can be “any name” but it must be saved as the Cygwin’s c:/"any name".bat Note that the file is saved with the suffix of “bat”.	The *.bat script is created.		This .bat file is the JTEL Search Script.	
<b>C. Execute the Search Script</b>				
11. Create the source file directory as the Cygwin’s c:/Source_Code Create a test results directory as the Cygwin c:/Search_Results.	The directories are created.			
12. Change to the Cygwin c directory. cd c:/				
13. Execute the batch file you have just created. ./"any name".bat NOTE: the leading dot-slash before the filename.				

APP_TC_024				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>D. Review the results of the Search Script for minimumCORBA violation. (AP0607)</b>				
14. Review the output file “minCORBA_Output_2.txt”, for minimum CORBA violations and record the violations in the Log.	<p><b>Pass:</b> The application only uses the CORBA constructs that are allowed by the minimumCORBA specification. (AP0607)</p> <p><b>Fail:</b> The application uses CORBA constructs that are not allowed by the minimumCORBA specification. (AP0607)</p>		<p>a. “minCORBA_Output.txt is the outputs of the keywords</p> <p>b. “minCORBA_Output_1.txt, which is a filtered file of commenting lines</p> <p>c. “minCORBA_Output_2.txt, which is a filtered file from output1</p>	
<b>End of Test</b>				
<b>Test Method 2</b>				
<b>A. Prepare the test environment for manual search.</b>				
1. Select a keyword search tool.			Any keyword search tool is acceptable.	
2. Set the folder directory to the application source files.				
<b>B. Execute keyword search using provided CORBA call list.</b>				
3. Open the “AP0607 Procedure” spreadsheet and navigate to the “minCORBA Violations” tab.				



APP_TC_024				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Copy Column A into a new worksheet or workbook Column A. Name worksheet with a name relating to the product.	A new worksheet or workbook is created with the data from minCORBA Violations tab.		Name of new worksheet or worksheet is not critical to the test. Ideally, this new worksheet or worksheet will be used for reporting.	
5. From the list of CORBA calls in the new worksheet/workbook, perform a keyword search.				
6. Log file(s) returned in Column B of new worksheet/workbook corresponding to the CORBA operation. If no file is returned, note "None" in the respective cell.				
<b>C. Review the search results for minimumCORBA violation. (AP0607)</b>				
7. Once all keywords have been searched, examine each file returned in Step 5 (shown in Column B) and confirm the violation(s) by reviewing the file.	<p><b>Pass:</b> The application only uses the CORBA constructs that are allowed by the minimumCORBA specification. (AP0607)</p> <p><b>Fail:</b> The application uses CORBA constructs that are not allowed by the minimumCORBA specification. (AP0607)</p>			
<b>End of Test</b>				

Test Recording Log – APP_TC_024	
minimum CORBA Violation	File Name(s)

## Test Summary APP\_TC\_024

Once testing is complete for every component of the application 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):**

AP0607 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.24APP\_TC\_025 – PropertySet::query()

### Test Case Number: APP\_TC\_025

PropertySet::query

query::UnknownProperties exception

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0095	The query operation shall return all component properties when the inout parameter configProperties is zero size.
AP0096	The query operation shall return only those id/value pairs specified in the configProperties parameter if the parameter is not zero size.
AP0097	Valid properties for the query operation shall be all configure properties (simple properties whose kind element's kindtype attribute is "configure") whose mode attribute is "readwrite" or "readonly" and any allocation properties with an action value of "external" as referenced in the component's SPD.
AP0098	The query operation shall raise the CF UnknownProperties exception when one or more properties being requested are not known by the component.

### References

Document Name	Version/Date	Location (page, section)
Software Communication Architecture (SCA)	Version 2.2.2 05-15-2006	3-13, 3-14, 3-94
SCA AppendixD – Domain Profile	Version 2.2.2 05-15-2006	D-19, D-21
SCA AppendixC – Core Framework IDL	Version 2.2.2 05-15-2006	C-8, C-15, C-16

### Test Objective

This test case verifies AP0095, AP0096, AP0097, and AP0098. The objective of this test is to verify the PropertySet query operation, which allows a component to be queried to return its properties. This test also verifies that the UnknownProperties exception is raised when one or more properties are not known by the component.

### Places to Verify

All instances of the PropertySet::query() operation within all of the application's components.

### IDL References

#### Exceptions

exception UnknownProperties {Properties invalidProperties; };

## Operations

void query (inout Properties configProperties) raises (UnknownProperties);

## Preconditions

- Domain Profile files are available and have passed the Domain Profile test (APP\_TC\_018).
- The application source code files are available.

## Test Description

For each application and application component having the query operation

- A. Locate the PRF file. (AP0097)
  1. Verify that the Software Assembly Descriptor (SAD) file exists
    - a. **Pass:** A SAD file exists.
    - b. **Fail:** A SAD file does not exist.
  2. Verify that a Software Package Descriptor (SPD) file is referenced in the SAD *componentfile*.
    - a. **Pass:** The SPD file exists.
    - b. **Fail:** The SPD file does not exist.
  3. For each SPD file under test, verify if a Properties Descriptor (PRF) file is referenced (AP0097)
    - a. **Untested:** A PRF file does not exist.
    - b. **Pass:** A PRF file exists and is referenced in the SPD file.
  4. For each SPD file under test, verify if a Software Component Descriptor (SCD) file is referenced (AP0097)
    - a. **Untested:** A SCD file does not exist.
    - b. **Pass:** A SCD file exists and is referenced in the SPD file.
  5. For each SCD file found in the previous step, verify if a Properties Descriptor (PRF) file is referenced (AP0097)
    - a. **Untested:** A PRF file does not exist.
    - b. **Pass:** A PRF file exists and is referenced in the SCD file.
- B. Verify that the query operation returns only readwrite and readonly configure properties with a kindtype of configure that are defined in the PRF. (AP0097)
  1. **Pass:** The query operation returns only readwrite and readonly configure properties
  2. **Fails:** The query operation does not returns only readwrite and readonly configure properties
- C. Verify that the query operation returns only allocation properties with an action value of external that are defined in the SPD. (AP0097)
  1. **Pass:** The query operation returns allocation properties with an action value of external.
  2. **Fails:** The query operation does not return allocation properties with an action value of external.

- D. Locate all instantiations of the query operation in the source code (AP0095, AP0096, AP0097, AP0098).
  - 1. **Pass:** The query operation is located in the source code.
  - 2. **Fail:** The query operation is not located in the source code.
- E. Verify that the query operation returns specified properties when the *configProperties* length size is not zero (AP0096).
  - 1. **Pass:** The query operation returns the specified properties.
  - 2. **Fail:** The query operation does not return the specified properties.
- F. Verify that the query operation returns all component properties when the *configProperties* length size is zero (AP0095).
  - 1. **Pass:** The query operation returns all properties.
  - 2. **Fail:** The query operation does not return all the properties.
- G. Verify that each query operation raises the *UnknownProperties* exception when one or more properties are not known by the component (AP0098).
  - 1. **Pass:** The *UnknownProperties* exception is raised when one or more properties are not known by the component
  - 2. **Fail:** The *UnknownProperties* exception is not raised when one or more properties are not known by the 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 it.

APP_TC_025				
Steps	Expected Results	Actual Results	Comments	Test Result
For each application and application component having the query operation:				
A. Locate the PRF file. (AP0097)				
1. Locate and examine the SAD file for its Software Package Descriptor (SPD) file.	<b>Pass:</b> A SAD file exists. (AP0097) <b>Fail:</b> A SAD file does not exist. (AP0097)		SPD declaration: <localfile name="xxxxx.spd.xml"/>	
2. Locate and examine the SPD file for its Properties Descriptor (PRF) file.	<b>Untested:</b> A PRF file does not exist. (AP0097) <b>Pass:</b> A PRF file exists and referenced in the SPD file. (AP0097)		<!ELEMENT propertyfile (localfile)>	
3. Locate and examine the SPD file for its Software Component Descriptor (SCD) file.	<b>Untested:</b> A SCD file does not exist. (AP0097) <b>Pass:</b> A SCD file exists. (AP0097)		SCD declaration: <localfile name="xxxxx.scd.xml"/>	
4. Locate and examine the SCD file for its Properties Descriptor (PRF) file.	<b>Untested:</b> A PRF file does not exist. (AP0097) <b>Pass:</b> A PRF file exists and referenced in the SCD file. (AP0097)		<!ELEMENT propertyfile (localfile)>	
B. Verify that the query returns all readwrite and readonly configure properties that are defined in the Properties Descriptor file. (AP0097)				

APP_TC_025				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Record the names of readwrite and readonly configure properties in the test log.			<i>kindtype</i> set to "configure" <i>mode</i> set to "readonly" <b>OR</b> <i>mode</i> set to "readwrite"  <simple id="p_name" type="string" name="p_name" <b>mode="readwrite"</b> > <description>processor</description> <value>x86</value> <kind <b>kindtype="configure"</b> />	
6. Verify that the query operation returns the readwrite and readonly configure properties.	<b>Pass:</b> The query operation returns readwrite and readonly configure properties. (AP0097)  <b>Fail:</b> The query operation does not return readwrite and readonly configure properties. (AP0097)			
<b>C. Verify that the query operation returns only allocation properties with an action value of external that are defined in the SPD. (AP0097)</b>				
7. Record the names of allocation properties with an action value of external in the test log.			<i>Action value</i> set to "external" is the SPD  <simple id="p_name" type="string" name="p_name" <b>mode="readwrite"</b> > <description>processor</description> <value>x86</value> <kind <b>kindtype="configure"</b> />	



APP_TC_025				
Steps	Expected Results	Actual Results	Comments	Test Result
8. Verify that the query operation returns the allocation properties.	<b>Pass:</b> The query operation returns allocation properties. (AP0097)  <b>Fail:</b> The query operation does not return allocation properties. (AP0097)			
<b>D. Locate all instantiations of the query operation in the source code. (AP0095, AP0096, AP0097, AP0098)</b>				
9. Search the source code for all occurrences of the implementation of the query operation and record the source file name.	<b>Fail:</b> The query operation is not implemented in the source code. (AP0095, AP0096, AP0097, AP0098)  <b>Pass:</b> The implementation of the query operation is located in the source code. (AP0095, AP0096, AP0097, AP0098)		<pre>void CF_Resource_i::query( CF::Properties&amp; props                     CF_ENV_ARG_DECL )</pre>	
<b>E. Verify that the query operation returns specified properties when the configProperties length size is not zero. (AP0096)</b>				
10. Verify that query operation performs checking of the input configProperties list to determine its length.	<b>Pass:</b> The query() operation verifies the input parameters length. (AP0096)  <b>Fail:</b> The query() operation does not verify the input parameters length. (AP0096)			
11. For each query() operation, verify that all component's properties in the input argument configProperties are returned, when the length of the configProperties is greater than zero.	<b>Pass:</b> The query operation returns all of the specified properties. (AP0096)  <b>Fail:</b> The query operation does not return all of the specified properties. (AP0096)		A non-zero length list contains a list of properties being asked for.	

APP_TC_025				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>F. Verify that the query returns all component properties when the configProperties length size is zero. (AP0095)</b>				
12. For each query operation, verify that all components' properties are returned, when the length of the configProperties is equal to zero.	<b>Pass:</b> The query operation returns all properties. (AP0095)  <b>Fail:</b> The query operation does not return all the properties. (AP0095)		A zero length list means all properties are being asked for.	
<b>G. Verify that the query operation raises the UnknownProperties exception when one or more properties are not known by the component. (AP0098)</b>				
13. For each query operation, verify that the UnknownProperties exception is raised when properties in the input argument are not known by the component.	<b>Pass:</b> The UnknownProperties exception is raised when one or more properties are not known by the component. (AP0098)  <b>Fail:</b> The UnknownProperties exception is not raised when one or more properties are not known by the component. (AP0098)		<pre>ptrpException = new CF::FileSystem:: UnknownFileSystemProperties (invalidProperties);</pre>	
<b>End of Test</b>				

Test Recording Log – APP_TC_025						
SAD File:						
Notes						
Step 2 & 4 (PRF file)	Step 5 (mode type)	Step 6 & 7 (returns allocation properties Y-N)	Step 10 (configProperties zero length-Y/N?)	Step 11 (id, value, range)	Step 12 (properties returned – Y/N?)	Step 13 (exception raised – Y/N?)

## Test Summary APP\_TC\_025

Once testing is complete for every component of the application 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):**

AP0095\_\_\_\_\_

AP0096\_\_\_\_\_

AP0097\_\_\_\_\_

AP0098\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.25APP\_TC\_026 – Base Application Interfaces

**Test Case Number:** APP\_TC\_026

CF::Base Application Interfaces

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0608	Applications shall implement the Base Application Interfaces as specified in section 3.1.3.1 using the corresponding IDL in Appendix C.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-95, Section 3.2.1.3
<i>SCA Appendix C – IDL</i>	Version 2.2.2 05-15-2006	Pages C-1, C-10, C11, C-23, C24, C-14, C-15, C16
<i>SCA Appendix D – Domain Profile</i>	Version 2.2.2 05-15-2006	Pages D-4 to D-13, Section D.2.1

### Test Objective

This test case verifies AP0608. The objective of this test is to verify that each component identified as implementing the Base Application Interfaces (Ports, LifeCycle, TestableObject, PropertySet, PortSupplier, Resource or ResourceFactory) follow the specifications identified in section 3.1.3.1 of the SCA. Furthermore, this test verifies that the interfaces implemented match the IDL signature in Appendix C.

### Places to Verify

Components implementing the Base Application Interfaces

### IDL References

#### Operations

```

interface Resource : LifeCycle, TestableObject, PropertySet, PortSupplier {
    void start () raises (CF::Resource::StartError);
    void stop () raises (CF::Resource::StopError);
    exception StartError { CF::ErrorNumberType errorNumber; string msg; };
    exception StopError { CF::ErrorNumberType errorNumber; string msg; };
};

interface LifeCycle {

```

```
void initialize () raises (CF::LifeCycle::InitializeError);
void releaseObject () raises (CF::LifeCycle::ReleaseError);
exception InitializeError { CF::StringSequence errorMessages; };
exception ReleaseError { CF::StringSequence errorMessages; };
};
```

**interface TestableObject {**

```
void runTest ( in unsigned long testid, inout CF::Properties testValues )
    raises (CF::TestableObject::UnknownTest, CF::UnknownProperties);
exception UnknownTest { };
```

```
};
```

**interface PropertySet {**

```
void configure ( in CF::Properties configProperties )
    raises (CF::PropertySet::InvalidConfiguration, CF::PropertySet::PartialConfiguration);
void query ( inout CF::Properties configProperties ) raises (CF::UnknownProperties);
exception InvalidConfiguration {string msg; CF::Properties invalidProperties; };
exception PartialConfiguration { CF::Properties invalidProperties; };
```

```
};
```

**interface PortSupplier {**

```
Object getPort ( in string name) raises (CF::PortSupplier::UnknownPort);
exception UnknownPort { };
```

```
};
```

**interface Port {**

```
void connectPort ( in Object connection, in string connectionId ) raises (CF::Port::InvalidPort,
CF::Port::OccupiedPort);
void disconnectPort ( in string connectionId ) raises (CF::Port::InvalidPort);
exception InvalidPort { unsigned short errorCode; string msg; };
exception OccupiedPort { };
```

```
};
```

**interface ResourceFactory{**

```
CF::Resource createResource ( in string resourceId, in CF::Properties qualifiers )
    raises (CF::ResourceFactory::CreateResourceFailure);
```

```
void releaseResource ( in string resourceId ) raises (CF::ResourceFactory::InvalidResourceId);  
void shutdown ( ) raises (CF::ResourceFactory::ShutdownFailure);  
exception InvalidResourceId { };  
exception ShutdownFailure {string msg; };  
exception CreateResourceFailure {CF::ErrorNumberType errorNumber; string msg; };  
};
```

## Preconditions

- The waveform CF IDL must be available.
- The SCA CF.idl is available.

## Test Description

- A. Verify that the waveform CF.idl matches the SCA's CF.idl, except for white space and comment differences. (AP0608)
  1. **Pass:** The waveform CF.idl matches the SCA's CF.idl.
  2. **Fail:** The waveform CF.idl does not match the SCA's CF.idl.
- B. If the component is designated as a Resource, verify that the component implements the *Resource* interface as well as all four of the Base Application interfaces: *LifeCycle*, *TestableObject*, *PortSupplier* and *PropertySet*. (AP0608)
  1. **Pass:** All of the interfaces, *LifeCycle*, *TestableObject*, *PortSupplier* and *PropertySet* are implemented by the component.
  2. **Fail:** One or more of the interfaces, *LifeCycle*, *TestableObject*, *PortSupplier*, *PropertySet* or *Resource* was not implemented by the 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 it.

APP_TC_026				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Verify that the waveform CF.idl matches the SCA's CF.idl, except for white space and comment differences. (AP0608)</b>				
1. Verify that the waveform and SCA CF IDL content match.	<b>Pass:</b> The waveformCF.idl matches the SCA's CF.idl. (AP0608)  <b>Fail:</b> The waveformCF.idl does not match the SCA's CF.idl. (AP0608)		A compilable CF.idl file may be found in the JTAP's software install at \ProgramFiles\JTEL\JTAP\SCA2.2.2IDL.	
<b>B. If the component is designated as a Resource, verify that the component implements the <i>Resource</i> interface as well as all the Base Application interfaces: <i>LifeCycle</i>, <i>TestableObject</i>, <i>PortSupplier</i> and <i>PropertySet</i>. (AP0608)</b>				
2. Verify that the waveform <i>Resource</i> component implementation(dot-h file) conforms to (inherits from) the compiled CF.idl.	<b>Pass:</b> The <i>Resource</i> interface is implemented by the component. (AP0608)  <b>Fail:</b> The <i>Resource</i> interface is not implemented by the component. (AP0608)		<pre> module CF {     interface Resource;      interface ResourceFactory     {         /** This exception indicates the resourceId does not exist in the ResourceFactory. */         exception InvalidResourceId         {         };     }; </pre>	
<b>Verify that the component implements the Base Application interface <i>LifeCycle</i>. (AP0608)</b>				



APP_TC_026				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify that the waveform <i>LifeCycle</i> component implementation(dot-h file) conforms to (inherits from) the compiled CF.idl.	<p><b>Pass:</b> <i>LifeCycle</i> interface is implemented by the component. (AP0608)</p> <p><b>Fail:</b> <i>LifeCycle</i> interface is not implemented by the component. (AP0608)</p>		<pre> interface LifeCycle {     exception StartError     {         CF::ErrorNumberType         errorNumber; string msg;     };      exception StopError     {         CF::ErrorNumberType         errorNumber; string msg;     };      readonly attribute string identifier;      void start() raises     (CF::Resource::StartError);      void stop() raises     (CF::Resource::StopError); }; </pre>	
Verify that the component implements the Base Application interface TestableObject interface. (AP0608)				
4. Verify that the waveform <i>TestableObject</i> component implementation(dot-h file) conforms to (inherits from) the compiled CF.idl.	<p><b>Pass:</b> <i>TestableObject</i> interface is implemented by the component. (AP0608)</p> <p><b>Fail:</b> <i>TestableObject</i> interface is not implemented by the component. (AP0608)</p>		<pre> interface TestableObject {     exception UnknownTest     {     };      void runTest(in unsigned long testId,     inout CF::Properties testValues )         raises     (CF::TestableObject::UnknownTest,     CF::UnknownProperties); }; </pre>	
Verify that the component implements the Base Application interface PortSupplier interface. (AP0608)				

APP_TC_026				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Verify that the waveform <i>PortSupplier</i> component implementation(dot-h file) conforms to (inherits from) the compiled CF.idl.	<b>Pass:</b> <i>PortSupplier</i> interface is implemented by the component. (AP0608)  <b>Fail:</b> <i>PortSupplier</i> interface is not implemented by the component. (AP0608)		<pre> interface PortSupplier {     exception UnknownPort     {     };      Object getPort( in string name )         raises         (CF::PortSupplier::UnknownPort); }; </pre>	
<b>Verify that the component implements the Base Application interface PropertySet interface. (AP0608)</b>				
6. Verify that the waveform <i>PropertySet</i> component implementation(dot-h file) conforms to (inherits from) the compiled CF.idl.	<b>Pass:</b> <i>PropertySet</i> interface is implemented by the component. (AP0608)  <b>Fail:</b> <i>PropertySet</i> interface is not implemented by the component. (AP0608)		<pre> interface PropertySet {     exception InvalidConfiguration     {         string msg;         CF::Properties invalidProperties;     };      exception PartialConfiguration     {         CF::Properties invalidProperties;     };      void configure( in CF::Properties         configProperties )         raises (CF::PropertySet::             InvalidConfiguration,             CF::PropertySet::             PartialConfiguration         );     void query( inout CF::Properties         configProperties ) </pre>	

APP_TC_026				
Steps	Expected Results	Actual Results	Comments	Test Result
			raises (CF::UnknownProperties); };	
<b>Verify that the component implements the Base Application interface Port interface. (AP0608)</b>				
7. Verify that the waveform <i>Port</i> component implementation(dot-h file) conforms to (inherits from) the compiled CF.idl.	<b>Pass:</b> <i>Port</i> interface is implemented by the component. (AP0608)  <b>Fail:</b> <i>Port</i> interface is not implemented by the component. (AP0608)		<pre> interface Port { void connectPort (     in Object connection,     in string connectionId )     raises (CF::Port::InvalidPort,            CF::Port::OccupiedPort);  void disconnectPort (     in string connectionId )     raises (CF::Port::InvalidPort);  exception InvalidPort {     unsigned short errorCode;     string msg; };  exception OccupiedPort { } };           </pre>	
<b>(OPTIONAL) Verify that the component implements the Base Application interface ResourceFactory. (AP0608)</b>				

APP_TC_026				
Steps	Expected Results	Actual Results	Comments	Test Result
8. Verify that the waveform <i>ResourceFactory</i> component implementation(dot-h file) conforms to (inherits from) the compiled CF.idl.	<p><b>Pass:</b> <i>ResourceFactory</i> interface is implemented by the component. (AP0608)</p> <p><b>Fail:</b> <i>ResourceFactory</i> interface is not implemented by the component. (AP0608)</p>		<pre> interface ResourceFactory {     exception InvalidResourceId { };     exception ShutdownFailure     {    string msg; };      exception CreateResourceFailure     {         CF::ErrorNumberType         errorNumber; string msg; };      readonly attribute string identifier;     CF::Resource CreateResource(         in string resourceId,         in CF::Properties qualifiers     ) raises     (CF::ResourceFactory::CreateResourc     eFailure);      void releaseResource(in string     resourceId ) raises     (CF::ResourceFactory::InvalidResour     ceId);      void shutdown()         raises     (CF::ResourceFactory::ShutdownFail     ure); }; </pre>	
End of Test				

Test Recording Log – APP_TC_026							
<b>Step 1</b> (waveformCF IDL matches SCA CF IDL content – Y/N?)	<b>Step 2</b> (Resource interface implemented – Y/N?)	<b>Step 3</b> (Lifecycle interface implemented – Y/N?)	<b>Step 4</b> (TestableObject interface implemented – Y/N?)	<b>Step 5</b> (PortSupplier interface implemented – Y/N?)	<b>Step 6</b> (PropertySet interface implemented – Y/N?)	<b>Step 7</b> (Port interface implemented – Y/N?)	<b>Step 8 (optional)</b> (ResourceFactory interface implemented – Y/N?)

## Test Summary APP\_TC\_026

Once testing is complete for every component of the application 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):**

AP0608\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.26APP\_TC\_027 – ResourceFactory::createResource

**Test Case Number:** APP\_TC\_027

ResourceFactory::createResource

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0748	The <i>createResource()</i> operation shall create a resource if no resource exists for the given <i>resourceId</i> and shall assign the given <i>resourceId</i> to a new resource.
AP0111	The <i>createResource()</i> operation shall return a reference to the created resource.
AP0706	If the resource already exists, the <i>createResource()</i> operation shall return a reference to the existing resource.
AP0753	The <i>createResource()</i> operation shall set a reference count to one, when the resource is initially created, or increment the reference count by one, when the resource already exists.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-18 through 3-19, Section 3.1.3.1.7.5.1
SCA AppendixD, Domain Profile	Version 2.2.2 05-15-2006	Page D-36

### Test Objective

This test case verifies AP0111, AP0706, AP0748 and AP0753. The objective of this test is to verify that the *ResourceFactory*'s *createResource()* operation will create a resource and not duplicate a resource. It verifies that the *createResource* operation sets the counter appropriately.

### Places to Verify

ResourceFactory and any other component implementing the createResource operation.

### IDL References

CF::Resource createResource (  
    in string resourceId,  
    in CF::Properties qualifiers

)  
raises (CF::ResourceFactory::CreateResourceFailure);

## Preconditions

- Domain Profile files are available and have passed the Domain Profile test (APP\_TC\_018).
- The application source code files are available.

## Test Description.

- A. Perform a search on the source code for the createResource operation. (AP0111, AP0706, AP0748, AP0753)
  1. **Pass:** createResource operation located.
  2. **Fail:** createResource operation not located.
- B. Verify that the *createResource()* operation creates a resource if no resource exists for the given *resourceId* and assigns the given *resourceId* to the new resource (AP0748).
  1. **Pass:** The *createResource()* operation creates a resource if no resource exists for the given *resourceId* and assigns the given *resourceId* to the new resource.
  2. **Fail:** The *createResource()* operation does not create a resource if no resource exists for the given *resourceId*.
  3. **Fail:** The *createResource()* operation does not assign the given *resourceId* to the new resource.
- C. Verify that the *createResource()* operation returns the object reference to the created resource (AP0111).
  1. **Pass:** The *createResource()* operation returns a *reference* to the created resource.
  2. **Fail:** The *createResource()* operation does not return a *reference* to the created resource.
- D. Verify that the *createResource()* operation returns the object *reference* if the resource already exist and the input qualifiers parameter is ignored (AP0706).
  1. **Pass:** The *createResource()* operation returns a *reference* for the resource and the input qualifiers parameter is ignored if it already exists.
  2. **Fail:** The *createResource()* operation does not return a *reference* for the resource and the input qualifiers parameter is ignored if it already exists.
- E. Verify that the *createResource()* operation sets a *reference* count to one, when the resource is initially created, or increment the *reference* count by one, when the resource already exists (AP0753).
  1. **Pass:** The *createResource()* operation sets a *reference* count to one, when the resource is initially created, or increment the *reference* count by one, when the resource already exists.
  2. **Fail:** The *createResource()* operation does not set a *reference* count to one, when the resource is initially created, or does **not** increment the *reference* count by one, when the resource already exists.



## 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.

APP_TC_027				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Perform a search on the source code for the createResource operation. (AP0111, AP0706, AP0748, AP0753)</b>				
1. Perform a search on the source code for the createResource operation. (AP0111, AP0706, AP0748, AP0753)	<b>Pass:</b> <i>createResource()</i> operation located.  <b>Fail:</b> <i>createResource()</i> operation not located.			
<b>B. Verify that the <i>createResource()</i> operation creates a resource if no resource exists for the given <i>resourceId</i> and assigns the given <i>resourceId</i> to the new resource (AP0748).</b>				
2. Examine the source code files and verify that the <i>createResource()</i> operation will create a resource if no resource exists for the given or passed in <i>resourceId</i> . Newly created resources are assigned the given <i>resourceId</i> .	<b>Pass:</b> The <i>createResource()</i> operation creates a resource if no resource exists for the given <i>resourceId</i> and assigns the given <i>resourceId</i> to the new resource. (AP0748)  <b>Fail:</b> The <i>createResource()</i> operation does not create a resource if no resource exists for the given <i>resourceId</i> . (AP0748)  <b>Fail:</b> The <i>createResource()</i> operation does not assign the given <i>resourceId</i> to the new resource. (AP0748)			
<b>C. Verify that the <i>createResource()</i> operation returns an object reference for each created resource (AP0111).</b>				

APP_TC_027				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Examine the source code files and verify that the <i>createResource()</i> operation returns an object <i>reference</i> for each created resource.	<p><b>Pass:</b> The <i>createResource()</i> operation returns a <i>reference</i> to the created resource. (AP0111)</p> <p><b>Fail:</b> The <i>createResource()</i> operation does not return a <i>reference</i> to the created resource. (AP0111)</p>			
<b>D. Verify that the <i>createResource()</i> operation returns an object <i>reference</i> and the input qualifiers parameter is ignored if the resource already exists (AP0706).</b>				
4. Examine the source code files and verify that the <i>createResource()</i> operation returns an object <i>reference</i> and the input qualifiers parameter is ignored if the resource already exists.	<p><b>Pass:</b> The <i>createResource()</i> operation returns a <i>reference</i> for the resource and the input qualifiers parameter is ignored if it already exists. (AP0706)</p> <p><b>Fail:</b> The <i>createResource()</i> operation does not return a <i>reference</i> for the resource and the input qualifiers parameter is ignored if it already exists. (AP0706)</p>			
<b>E. Verify that the <i>createResource()</i> operation sets a <i>reference</i> count to one, when the resource is initially created, or increment the <i>reference</i> count by one, when the resource already exists (AP0753).</b>				

APP_TC_027				
Steps	Expected Results	Actual Results	Comments	Test Result
5. Examine the source code files and verify that the <i>createResource()</i> operation sets a <i>reference</i> count to one, when the resource is initially created, or increment the <i>reference</i> count by one, when more references to this resource are requested.	<b>Pass:</b> The <i>createResource()</i> operation sets a <i>reference</i> count to one, when the resource is initially created, or increment the <i>reference</i> count by one, when the resource already exists. (AP0753)  <b>Fail:</b> The <i>createResource()</i> operation does not set a <i>reference</i> count to one, when the resource is initially created, or does not increment the <i>reference</i> count by one, when the resource already exists. (AP0753)			
End of Test				

Test Recording Log – APP_TC_027				
Step 1 Source Code File	Step 2 Resource created	Step 3 Object references returned	Step 4 No new resource created if it already exists	Step 5 Reference count

**Test Summary APP\_TC\_027**

Once testing is complete for every component of the application 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):**

AP0748 \_\_\_\_\_

AP0753 \_\_\_\_\_

AP0111 \_\_\_\_\_

AP0706 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.27 APP\_TC\_028 – Core Framework Interfaces

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

CF::Interfaces

### Requirements

<i>SCA v2.2.2 Tag</i>	<i>SCA v2.2.2 Text</i>
AP0609	Each application component shall support the mandatory Naming Context IOR, Name Binding, and the identifier execute parameters as described in 3.1.3.2.2.5.1, in addition to their user-defined execute properties in the component's SPD.
AP0610	Each application component shall bind its object reference to the Naming Context IOR using the Name Binding parameter.
AP0611	Each executable component of an application shall set its identifier attribute using the component identifier execute parameter.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	v2.2.2 05-15-2006	Page 3-95, Section 3.2.1 Page 3-28, Section 3.1.3.2.2.5.1.3

### Test Objective

This test case verifies AP0609, AP0610 and AP0611. The objective of this test is to verify all application components support the mandatory execute parameters and it uses the Name Binding parameter to bind its object reference to the Naming Context IOR. During an application instantiation, the `ApplicationFactory::create()` must include the mandatory execute parameters, Naming Context IOR, Name Binding, and Component Identifier. Therefore, all application components must support these parameters as defined in the SCA v2.2.2 specification section 3.1.3.2.2.5.1. Additionally, this test verifies that the application executable(s) identifier attribute is the component identifier execute parameter.

### Places to Verify

`ApplicationFactory` and any other component implementing the create operation.

### IDL References

```
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

- Domain Profile files are available and have passed the Domain Profile test (APP\_TC\_018).
- The application source code files are available.

## Test Description

- A. Identify all resources (application components) of the application.
- B. Build a list of executables that are registered with the Naming Service for each resource from Step A.
- C. For each resource and its executables found, verify that each resource accepts the Naming Context IOR, Name Binding, and the identifier execute parameters. (AP0609)
  1. **Pass:** The parameters are accepted.
  2. **Fail:** One or more resource does not accept one or more of the following parameters: NAMING\_CONTEXT\_IOR, NAME\_BINDING, COMPONENT\_IDENTIFIER.
- D. Verify that each resource binds its object reference to the naming context IOR using the Name Binding parameter. (AP0610)
  1. **Pass:** The resource binds its object reference to the naming context IOR using the Name Binding parameter.
  2. **Fail:** The resource does not bind its object reference to the naming context IOR using the Name Binding parameter.
- E. Verify that each resource executable sets its identifier attribute using the component identifier execute parameter. (AP0611)
  1. **Pass:** The identifier attribute is set to the identifier execute parameter.
  2. **Fail:** The identifier attribute is not equal to the component identifier execute parameter

## Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, N/A, or any other as appropriate.  
2. The Test Recording Log is intended to record data for each step that requires it.

APP_TC_028				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Identify all resources (application components) of the application.</b>				
1. Parse the SAD file for resources and their SPD references.	<p>The resources are listed as:</p> <pre>-&lt;componentfiles&gt;   &lt;componentfile id="xxx"&gt;</pre> <p>The SPD reference is below the componentfile id</p> <pre>&lt;localfile name="xxx.spd.xml"&gt;</pre>		Use the Test Recording Log at the end of the manual test steps to record additional test results.	
<b>B. Build a list of executables that are registered with the Naming Service for each resource from Step A.</b>				
<p>2. Locate in the SAD the <i>componentfilerefrefid</i> that matches each <i>componentfile id</i> from Step 1.</p> <p>Record the <i>componentfilerefrefid</i> and the <i>name</i> registered to the Naming Service.</p> <p>The “<b>name</b>” found here is used in relationship to “NAME_BINDING” below.</p>	<p>The componentfile refid is located in the SAD &lt;partitioning&gt; element:</p> <pre>&lt;partitioning&gt;   &lt;componentplacement&gt;     &lt;componentfileref refid="xxx"/&gt;     &lt;componentinstantiation id="..."&gt;</pre> <p>The name registered to the Naming Service is located in the SAD &lt;findcomponent&gt; element of the &lt;partitioning&gt; element:</p> <pre>&lt;findcomponent&gt;   &lt;naming service name="yyy"/&gt;</pre>			
<b>C. For each resource and its executables found, verify that each resource accepts the Naming Context IOR, Name Binding, and the identifier execute parameters. (AP0609)</b>				



APP_TC_028				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Use the list of executables collected in Step 2 to locate the source code for each resource. (See comments to the right.)	Source code can be located		Finding where, in the source code, that this requirement is satisfied is implementation-dependent. The tester may find that this requirement has been satisfied in one file, at the super class level. On the other hand, the tester may find the source code in multiple files, at the subclass level.	
4. Parse the code for the following strings to show that the entry point for this resource has code to accept these parameters:  “NAMING_CONTEXT_IOR”, “NAME_BINDING”, and “COMPONENT_IDENTIFIER” parameters.	<p><b>Pass:</b> The parameters are accepted. (AP0609)</p> <p><b>Fail:</b> No resources accept one or more of the following parameters: NAMING_CONTEXT_IOR, NAME_BINDING, COMPONENT_IDENTIFIER. (AP0609)</p>			
<b>D. Verify that each resource binds its object reference to the naming context IOR using the Name Binding parameter. (AP0610)</b>				
5. Locate where the resource binds itself to the naming service and verify that the naming context parameter is constructed properly for the bind or rebind operation.	<p><b>Pass:</b> The resource binds its object reference to the naming context IOR using the Name Binding parameter. (AP0610)</p> <p><b>Fail:</b> The resource does not bind its object reference to the naming context IOR using the Name Binding parameter. (AP0610)</p>		<p>For a discussion of how an object reference binds to the Naming Context IOR see SCA section 3.1.3.2.2.5.1.3.</p> <p>Searching the source code for "NAME_BINDING" may be a good starting point for this step.</p>	

APP_TC_028				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>E. Verify each resource sets its identifier attribute using the COMPONENT_IDENTIFIER execute parameter. (AP0611)</b>				
6. Parse the source code for the COMPONENT_IDENTIFIER parameter.	<b>Pass:</b> The identifier attribute is set to the identifier execute parameter. (AP0611)  <b>Fail:</b> The identifier attribute is not equal to the component identifier execute parameter. (AP0611)			
<b>End of Test</b>				

Test Recording Log – APP_TC_028							
SAD File:							
refid	namingservice		executable name	entrypoint	Code result	Bind result	Parse result
Step 2			Step 3		Step 4	Step 5	Step 6

## Test Summary APP\_TC\_028

Once testing is complete for every component of the application 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):**

AP0609\_\_\_\_\_

AP0610 \_\_\_\_\_

AP0611\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.28APP\_TC\_029 – Producer and Consumer of Events

### Test Case Number: APP\_TC\_029

Event Producer / Event Consumer

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0063	A component (e.g., Resource, DomainManager, etc.) that consumes events shall implement the CosEventCommPushConsumer interface.
AP0064	A component (e.g., Resource, Device, DomainManager, etc.) that produces events shall implement the CosEventCommPushSupplier interface and use the CosEventCommPushConsumer interface for generating the events.
AP0065	A producer component shall not forward or raise any exceptions when the connection to a CosEventCommPushConsumer is a nil or invalid reference.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-2, 3-14

### Test Objective

This test case verifies AP0063, AP0064, and AP0065. The objective of this test is to verify that if the component consumes or produces events, it must follow the respective interface(s) from CosEventComm.

### Places to Verify

Application Resources that utilize the OMG Event Service.

### IDL References

None.

### Preconditions

- The application source code files are available.

## Test Description

- A. Find each component that declares itself an event Consumer and verify that the CosEventComm *PushConsumer* interface is implemented. (AP0063)
  - 1. **N/A:** No component declares itself an event Consumer.
  - 2. **Fail:** An event Consumer component does not implement the CosEventComm *PushConsumer* interface
  - 3. **Pass:** The event Consumer components implement the CosEventComm *PushConsumer* interface.
- B. Find each component that declares itself an event Supplier and verify that the CosEventComm *PushSupplier* is implemented. (AP0064)
  - 1. **N/A:** No component declares itself an event Supplier.
  - 2. **Fail:** An event Supplier component does not implement the CosEventComm *PushSupplier* interface
  - 3. **Pass:** The event Supplier components implement the CosEventComm *PushSupplier* interface.
- C. Verify that the components found in step B use the CosEventComm *PushConsumer* interface for generating the events.
  - 1. **Fail:** The event producer does not use a consumer's *PushConsumer* interface for generating events.
  - 2. **Pass:** The event producer uses a consumer's *PushConsumer* interface for generating events.
- D. Verify that the components found in step B capture or handle exceptions without forwarding when a NIL or invalid reference is used for the connection to the *PushConsumer*. (AP0065)
  - 1. **Fail:** The event producer fails to capture or handle the exception due to an invalid or NIL *PushConsumer* reference.
  - 2. **Fail:** The exception handling, due to an invalid or NIL *PushConsumer* reference, results in a further propagation of this (or another) exception.
  - 3. **Pass:** The Event Producer component internally handles all exceptions related to NIL or invalid CosEventComm::*PushConsumer* references and does not propagate them outside the 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 it.

APP_TC_029				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Find each component that declares itself an event <i>Consumer</i> and verify that the CosEventComm <i>PushConsumer</i> interface is implemented. (AP0063)</b>				
1. Search the source code for an event <i>Consumer</i> .	<b>N/A:</b> No component declares itself an event <i>Consumer</i> . (AP0063)  Skip step 2.		Consult the software developer to locate these files.  A keyword search on “PushConsumer” can be a good starting point.	
2. Verify that the component implements the push and disconnect_push_consumer methods.	<b>Fail:</b> The <i>push</i> or the <i>disconnect_push_consumer</i> method is not implemented (AP0063) <b>Fail:</b> The pull and the <i>disconnect_pull_consumer</i> method are implemented. (AP0063) <b>Pass:</b> The push and the <i>disconnect_push_consumer</i> method are implemented. (AP0063)		Implementation of any pull interface methods constitutes a failure.	
<b>B. Find each component that declares itself an event <i>Supplier</i> and verify that the CosEventComm <i>PushSupplier</i> is implemented. (AP0064)</b>				
3. Search the source code for <i>PushSupplier</i> .	<b>N/A:</b> No component declares itself a <i>PushSupplier</i> . (AP0064)  Skip step 4.		A keyword search on “PushSupplier” can be a good starting point.	

APP_TC_029				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Verify that the component implements the <code>disconnect_push_supplier(..)</code> methods.	<p><b>Pass:</b> The <code>disconnect_push_supplier</code> method is implemented. (AP0064)</p> <p><b>Fail:</b> The <code>disconnect_pull_supplier</code> method is implemented. (AP0064)</p> <p><b>Fail:</b> The <code>disconnect_push_supplier</code> method is not implemented (AP0064)</p>		Implementation of any pull interface methods constitutes a failure.	
<b>C. Verify that the components found in step B use the CosEventComm PushConsumer interface for generating the events. (AP0064)</b>				
5. Verify that whenever the producer sends events, it calls the consumer's <code>PushConsumer::push()</code> .	<p><b>Pass:</b> The event is sent to the consumer by calling the consumer's <code>PushConsumer::push()</code> function. (AP0064)</p> <p><b>Fail:</b> The event is sent to the consumer without calling the consumer's <code>PushConsumer::push()</code> function. (AP0064)</p>		The producer may have a function that specifically handles event production, or it may simply invoke the <code>PushConsumer::push()</code> function inside other functions whenever it needs to produce the events. Consult with developers when in doubt.	
<b>D. Verify that the components identified in step B capture or handle exceptions without forwarding when a NIL or invalid reference is used for the connection to the PushConsumer. (AP0065)</b>				
6. Locate all invocations of the <code>PushConsumer::push</code> method in the source code.				



APP_TC_029				
Steps	Expected Results	Actual Results	Comments	Test Result
7. Verify that when the PushConsumer object is a nil or invalid reference, the producer component will not raise or propagate a subsequent exception.	<b>Pass:</b> No exception is raised or propagated if the <i>PushConsumer</i> object is a nil or an invalid reference. (AP0065)  <b>Fail:</b> An exception is raised or can be propagated outside of the producer component if the <i>pushConsumer</i> object is nil or invalid. (AP0065)		Ensure either one of the following: <ol style="list-style-type: none"><li>1. There is error checking on the PushConsumer object's validity before making calls to it.</li><li>2. Any reference on the PushConsumer object is placed within exception handling block (try-catch as in C++), and any potential exceptions caused by nil or invalid object reference are to be suppressed.</li></ol>	
End of Test				

Test Recording Log – APP_TC_029					
<b>Step 1</b> ( <i>PushConsumer files</i> )	<b>Step 2</b> ( <i>push &amp; disconnect_push_consumer</i> )	<b>Step 3</b> ( <i>PushSupplier files</i> )	<b>Step 4</b> ( <i>disconnect_push_supplier</i> )	<b>Step 5</b> ( <i>push invocations</i> )	<b>Step 6</b> ( <i>exception handling</i> )

## Test Summary APP\_TC\_029

Once testing is complete for every component of the application 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):**

AP0063\_\_\_\_\_

AP0064\_\_\_\_\_

AP0065\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.29 APP\_TC\_030 – AEP, POSIX::kill()

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

*kill()*

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0662	An application that conforms to the AEP shall not call the <i>kill()</i> function with a negative argument because this profile does not require process group functionality.

### References

Document Name	Version/Date	Location (page, section)
SCA	Version 2.2.2 05-15-2006	Pages 1-4, 3-1
SCA Appendix B	Version 2.2.2 05-15-2006	Pages B-4, B-5

### Test Objective

This test verifies AP0662. The objective of this test is to verify that an application calls the POSIX *kill* function properly (i.e., without negative arguments).

### Preconditions

- The application source code files are available.

### Test Description

- A. Examine the source code of each application that calls the *kill()* operation (AP0662)
  - a. **Pass:** A negative argument is not passed to the *kill()* function
  - b. **Fail:** A negative argument is passed to the *kill()* function

## 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.

APP_TC_030				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Examine each application that calls the <i>kill()</i> operation that a negative argument cannot be passed</b>				
1. Examine the source code of each call to the <i>kill()</i> operation.	Write list of applications having the <i>kill()</i> operation into the Test Recording Log Sheet.  <b>N/A</b> : Cannot locate any <i>kill()</i> operations. Test ends since applications are not required to use the <i>kill()</i> operation.		<i>kill</i> <signal_number> <pid>	
2. Verify that a negative argument is not passed into the <i>kill()</i> operation	<b>Pass</b> : Negative arguments are not passed into the <i>kill()</i> operand  <b>Fail</b> : Negative arguments are passed into the <i>kill()</i> operand.		<signal_number> is a non-negative decimal integer	
<b>End of Test</b>				

Test Recording Log – APP_TC_030	
Step 1 (applications having <i>kill()</i> operation)	

**Test Summary APP\_TC\_030**

Once testing is complete for every component of the application 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):**

AP0662 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.30APP\_TC\_031 – ResourceFactory::Identifier

**Test Case Number:** APP\_TC\_031

Resource Factory::identifier

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0705	The readonly identifier attribute shall contain the unique identifier for a ResourceFactory instance.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-18, Section 3.1.3.1.7. Pages 4-2, Section 4.5
SCA AppendixD: Domain Profile	Version 2.2.2 FINAL / 05-15-2006	Page D-4, Section D.2.1 Page D-7, Section D.2.1.6 Page D-36, Section D.6.1.3.3

### Test Objective

This test case verifies AP0705. The objective of this test is to verify that the readonly identifier attribute contains the unique identifier for a ResourceFactory instance.

### Places to Verify

ResourceFactory source code implementation.

### IDL References

None

### Preconditions

- Domain Profile files are available and have passed the Domain Profile test (APP\_TC\_018).
- The application source code files are available.

### Test Description

A. Find the set of the resource factories for the application.



1) **N/A**: There are no resource factories in the application. End Test.

For each resource factory found perform the following;

B. Verify that the resourceFactory's identifier attribute is in the DCE format, indicating uniqueness. (AP0705)

1) **Fail**: The resourceFactory's identifier is not in the DCE format.

2) **Pass**: The resourceFactory's identifier is in the DCE format, indicating uniqueness.

C. Verify that the identifier attribute cannot be modified.(AP0705)

1) **Fail**: The resourceFactory identifier can be modified.

2) **Pass**: The resourceFactory identifier is unable to be modified.

## 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.

APP_TC_031				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Find the set of the resource factories for the application.</b>				
1. Find the set of the resource factories.	N/A: There are no resource factories in the application. End the Test. (AP0705)			
<b>NOTE:</b> The application software engineer can be of assistance in locating the source code for the resource factory(s). If the application software engineer is not available, use a simple search tool. A search engine such as grep or a development tool such as VisualC++ can be used to search all the source code to find the operation. Do not depend on Microsoft Explorer's search operation, as it can be demonstrated that it will not find a string within a file. Note: the executable name is a clue to the source code. Conventional practice is to have the same name for the executable as the primary source code file.				
<b>For each Resource Factory found perform the following:</b>				
<b>B. Verify that the resourceFactory's identifier attribute is in the DCE format, indicating uniqueness. (AP0705)</b>				
2. Examine the source code and verify that the ResourceFactory identifier is set to a value that is in DCE format	<b>Pass:</b> The resourceFactory's identifier is in the DCE format, indicating uniqueness. (AP0705)  <b>Fail:</b> The resourceFactory's identifier is not in the DCE format. (AP0705)		Example:  "DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1"	
<b>C. Verify that the identifier attribute cannot be modified. (AP0705)</b>				

APP_TC_031				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify in the sourcecode that the resourceFactory's identifier attribute is only assigned when it is instantiated and not modified or modifiable elsewhere in the code.	<b>Pass:</b> The resourceFactory identifier is unable to be modified. (AP0705)  <b>Fail:</b> The resourceFactory identifier can be modified. (AP0705)		<b>NOTE:</b> The readonly quality may be determined in the following ways. <ul style="list-style-type: none"><li>▪ declaring the identifier as static</li><li>▪ In the IDL, use the word readonly in front of the attribute name.</li></ul> The class file that contains the resourceFactory sourcecode should not contain any "setter" functions for the resourceFactory identifier.	
End of Test				

Test Recording Log – APP_TC_031		
Step 1 (ResourceFactories)	Step 2 (resourceFactory identifier in DCE format – Y/N?)	Step 3 (resourceFactory identifier able to be modified-Y/N?)

## Test Summary APP\_TC\_031

Once testing is complete for every component of the application 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):**

AP0705 \_\_\_\_\_

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

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

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

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

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

## B.31APP\_TC\_032 – Resource::start() raises StartError

**Test Case Number:** APP\_TC\_032

Resource::start() raises StartError

### Requirements

SCA v2.2.2. Tags	2.2.2. Requirement Text
AP0105	The <i>start</i> operation shall raise the StartError exception if an error occurs while starting the resource.
AP0099	The errorNumber parameter shall indicate a CF ErrorNumberType value.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-16, Section 3.1.3.1.6.3.1 and Section 3.1.3.1.6.5.1.5 Page 3-94, Section 3.1.3.6.13
SCA AppendixD	Version 2.2.2 05-15-2006	Pages D1 – D13, D19 – D26, D28 – D31

### Test Objective

This test verifies AP0099 and AP0105. The objective of this test is to verify the existence of a resource's *start* operation and that it handles exceptions, promulgates the exception messages, and provides an error number of type CF::ErrorNumberType.

### Places to Verify

Applications or components that require maintaining client applications using the Resource interface.

### IDL References

#### Data

```
enum ErrorNumberType {  
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,  
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,  
    CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_MSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,  
    CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,
```

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

### Exceptions

exception StartError { CF::ErrorNumberType errorNumber; string msg; };

### Operations

void start ()  
    raises (CF::Resource::StartError);

### Preconditions

- The application source code files are available.

### Test Description

- A. Search the source code for the *start* function. (AP0099, AP0105)
  1. **Fail:** No *start* function exists. If there is no *start()* function then the remaining requirements, AP099 and AP0105, will be untested. Continue with next SPD reference.
- B. Verify that the *start* function raises StartError. This may involve tracing code to ensure that errors are propagated back to the start function. (AP0105)
  1. **Fail:** StartError not raised or an error not propagated back to start function. If there is no StartError raised then the remaining requirement, AP0099, will be untested.
- C. Verify that the *StartError* exception error number is a CF *ErrorNumberType* as defined in Appendix C of the SCA. (AP0099)
  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.

APP_TC_032				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Search the source code for the <i>start</i> function.(AP0099, AP0105)</b>				
1. Locate and parse the corresponding source code of each executable source file for the <i>start</i> function.	<i>start</i> function should exist. (AP0099, AP0105)  <b>Fail</b> : <i>start</i> function does not exist. (AP0099, AP0105)		The <i>start</i> function should look similar to: <i>start()</i> .	
<b>B. Verify that the <i>start</i> function raises <i>StartError</i>. This may involve tracing code to ensure that errors are propagated back to the <i>start</i> function. (AP0105)</b>				
2. Review each defined <i>start()</i> operation to ensure that it raises the <i>StartError</i> exception if an error occurs while starting the resource.	<b>Pass</b> : <i>StartError</i> exception is raised when the <i>start</i> function fails to start a resource. (AP0105)  <b>Fail</b> : <i>StartError</i> exception is not raised when the <i>start</i> function fails to start a resource. (AP0105)		The <i>StartError</i> exception should contain the word: <i>StartError</i>	
<b>C. Verify that the <i>StartError</i> exception error number is a CF <i>ErrorNumberType</i> as defined in Appendix C of the SCA. (AP0099)</b>				
3. Verify that the <i>StartError</i> exception error number is a CF <i>ErrorNumberType</i> as defined in Appendix C of the SCA	<b>Pass</b> : The exception includes an error number of <i>ErrorNumberType</i> . (AP0099)  <b>Fail</b> : The exception does not include an error number of <i>ErrorNumberType</i> . (AP0099)		The error number values can be found in Appendix C, Pages C-3 and C-4. They are listed in the IDL section above.	
<b>End of Test</b>				

### Test Recording Log – APP\_TC\_032



SAD File:			
Step 1 Source Codes for <i>start()</i>	Step 2 Correct exception raised	Step 3 CF ErrorNumberType	Notes

## Test Summary APP\_TC\_032

Once testing is complete for every component of the application 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):**

AP0099 \_\_\_\_\_

AP0105 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.32APP\_TC\_033 – Resource::stop raises StopError

**Test Case Number:** APP\_TC\_033

Resource::stop()

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0100	The <i>errorNumber</i> parameter shall indicate a CF <i>ErrorNumberType</i> value.
AP0102	The <i>stop</i> operation shall disable all current operations and put the resource in a non-operating condition.
AP0103	The <i>stop</i> operation shall raise the <i>StopError</i> exception if an error occurs while stopping the resource.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-16– 3-17, Section 3.1.3.1.6.3.1 and Section 3.1.3.1.6.5.2.5 Page 3-94, Section 3.1.3.6.13
SCA AppendixD	Version 2.2.2 05-15-2006	Pages D-42 to D-44

### Test Objective

This test case verifies AP0100, AP0102, and AP0103. The objective of this test procedure is to verify the existence of a stop operation for each resource. This test case also verifies that the stop operation disables all current operations, puts the resource in a suspended state, and handles an exception, providing an error number of type CF::ErrorNumberType.

### Places to Verify

Applications or components that require maintaining client applications using the Resource interface.

### IDL References

#### Data

```
enum ErrorNumberType {  
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,  
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,  
    CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_MSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,  
    CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,
```

---

```
CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS,  
CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };
```

### Exceptions

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

### Operations

```
void stop ()  
    raises (CF::Resource::StopError);
```

### Preconditions

- The application source code files are available.
- Vendor design specification documentation is available

### Test Description

For each resource:

- A. Locate the source code and verify the *stop()* operation exists. (AP0102)
  1. **Pass:** The *stop()* operation is found.
  2. **Fail:** The *stop()* operation is not found.
- B. Verify by examining the *stop()* operation that the Resource is placed into a non-operational condition. The JTR developer should have a definition for a non-operational state. (AP0102)
  1. **Pass:** The Resource is placed into a non-operational condition.
  2. **Fail:** The Resource is not placed into a non-operational condition.
- C. By examining the *stop()* operation, verify that the *StopError* exception is raised if an error occurs while stopping the resource. (AP0103)
  1. **Pass:** The *StopError* exception is raised for each error condition.
  2. **Fail:** The *stop()* operation raises or propagates exceptions other than or in addition to the *StopError* exception when an error occurs.
- D. Verify that the *StopError* exception error number is a CF *ErrorNumberType* as defined in Appendix C of the SCA. (AP0100)
  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.

APP_TC_033				
Steps	Expected Results	Actual Results	Comments	Test Result
<i>For each resource:</i>				
<b>A. Locate the source code and verify the stop() operation exists. (AP0102)</b>				
1. Locate and record the name of the source code for the <i>stop()</i> operations for each resource.	<b>Pass:</b> The <i>stop()</i> operation is found. (AP0102)  <b>Fail:</b> The <i>stop()</i> operation is not found. (AP0102)			
<b>B. Verify by examining the stop() operation that the Resource is placed into non-operational condition per the Vendor's documentation. All internal processing should stop. (AP0102)</b>				
2. Verify that the stop() operation puts the Resource into a non-operational condition. The resource stops servicing client requests.	<b>Pass:</b> The Resource is placed into a non-operational condition. (AP0102)  <b>Fail:</b> The Resource is not placed into a non-operational condition. (AP0102)		The design documents should describe the non-operating conditions of the resource.	
<b>C. Verify by examining the stop() operation, that the StopError exception is raised if an error occurs while stopping the resource. (AP0103)</b>				

APP_TC_033				
Steps	Expected Results	Actual Results	Comments	Test Result
3. Verify that the <i>StopError</i> exception is raised if an error occurs while stopping the resource. The <i>stop()</i> operation handles internal exceptions.	<p><b>Pass:</b> The <i>StopError</i> exception is raised for each error condition. (AP0103)</p> <p><b>Fail:</b> The <i>stop()</i> operation raises or propagates exceptions other than or in addition to the <i>StopError</i> exception when an error occurs. (AP0103)</p>			
<b>D. Verify that the <i>StopError</i> exception error number is a CF <i>ErrorNumberType</i> as defined in Appendix C of the SCA. (AP0100)</b>				
4. Verify that the <i>StopError</i> exception error number is a CF <i>ErrorNumberType</i> as defined in Appendix C of the SCA	<p><b>Pass:</b> The exception includes an error number of <i>ErrorNumberType</i>. (AP0100)</p> <p><b>Fail:</b> The exception does not include an error number of <i>ErrorNumberType</i>. (AP0100)</p>		The error number values can be found in Appendix C, Pages C-3 and C-4. They are listed in the IDL section above.	
<b>End of Test</b>				

Test Recording Log -- APP_TC_033				
SADFile:				
Step 1	Step 2	Step 3	Step 4	Notes
Source code found	stop operates properly	<i>StopError</i> exception is generated properly	Valid error code	

## Test Summary APP\_TC\_033

Once testing is complete for every component of the application 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):**

AP0100 \_\_\_\_\_

AP0102 \_\_\_\_\_

AP0103 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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



## B.33APP\_TC\_034 – Resource::stop() not permanent

### Test Case Number: APP\_TC\_034

Resource::stop()

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0704	The stop operation shall not inhibit subsequent configure, query, and start operations.

### References

Document Name	Version/Date	Location
SCA	Version 2.2.2 05-15-2006	Pages 3-17, 3-18 – 3-20
SCA AppendixD	Version 2.2.2 05-15-2006	Pages D1 – D13, D19 – D26, D28 – D31

### Test Objective

This test case verifies requirement AP0704. The objective of this test is to verify that the resource's *stop()* operation does not inhibit subsequent calls to *configure()*, *query()*, and *start()*.

### Places to Verify

Applications or components that require maintaining client applications using the Resource interface.

### IDL References

#### Exceptions

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

#### Operations

```
void stop ()  
raises (CF::Resource::StopError);
```

### Preconditions

- The source code files are available.

## Test Description

Use the application SAD file to find all of its SPD files.

For each component:

- A. Verify that after a successful *stop()* operation, that the *query()* call can be fully executed. (AP0704)
  - 1. **Fail:** the *query()* call does not operate correctly following a successful *stop()* operation.
  - 2. **Pass:** the *query()* call operates correctly following a successful *stop()* operation.
- B. Verify that after a successful *stop()* operation, that the *configure()* call can be fully executed. (AP0704)
  - 1. **Fail:** The *configure()* call does not operate correctly following a successful *stop()* operation.
  - 2. **Pass:** The *configure()* call operates correctly following a successful *stop()* operation.
- C. Verify that after a successful *stop()* operation, that the *start()* call can be fully executed. (AP0704)
  - 1. **Fail:** The *start()* call does not operate correctly following a successful *stop()* operation.
  - 2. **Pass:** The *start()* call operates correctly following a successful *stop()* operation.

## Manual Test Steps

Note: The Test Recording Log is intended to record long comments and lists of SPD files, SCD files, PRF, and other file/data.

APP_TC_034				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Verify that after a successful <i>stop()</i> operation, the <i>query()</i> operation can be fully executed. (AP0704)</b>				
1. Examine the source code to verify the <i>query()</i> operation exists and the inheritance is implemented properly to avoid functionality override.	<p><b>Pass:</b> The <i>query()</i> operation exists with correct inheritance implementation. (AP0704)</p> <p><b>Fail:</b> The <i>query()</i> operation does not exist with correct inheritance implementation. (AP0704)</p>		Inheritance is implementation-specific. The tester may find that this requirement has been satisfied in one file, at the super class level. Or, the tester may find the source code in multiple files, at the subclass level.	
2. Examine the source code to verify that the <i>query()</i> operation can be executed if the application is in a stopped state.	<p><b>Pass:</b> The <i>query()</i> operation is executable if the application is in a stopped state. (AP0704)</p> <p><b>Fail:</b> The <i>query()</i> operation is not executable if the application is in a stopped state. (AP0704)</p>			
<b>B. Verify that after a successful <i>stop()</i> operation, that the <i>configure()</i> function can be fully executed. (AP0704)</b>				
3. Examine the source code to verify the <i>configure()</i> operation exists and the inheritance is implemented properly to avoid functionality override.	<p><b>Pass:</b> The <i>configure()</i> operation exists with correct inheritance implementation. (AP0704)</p> <p><b>Fail:</b> The <i>configure()</i> operation does not exist with correct inheritance implementation. (AP0704)</p>		Inheritance is implementation-specific. The tester may find that this requirement has been satisfied in one file, at the super class level. Or, the tester may find the source code in multiple files, at the subclass level.	

APP_TC_034				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Examine the sourcecode to verify that the <i>configure()</i> operation continues to be processed, if the application is in a stopped state.	<p><b>Pass:</b> The <i>configure()</i> operation must be allowed to process, if the application is in a stopped state. (AP0704)</p> <p><b>Fail:</b> The <i>configure()</i> operation is not allowed to process, if the application is in a stopped state. (AP0704)</p>			
<b>C. Verify that after a successful <i>stop()</i> operation, that the <i>start()</i> function can be fully executed. (AP0704)</b>				
5. Examine the sourcecode to verify the <i>start()</i> operation exists and the inheritance is implemented properly to avoid functionality override.	<p><b>Pass:</b> The <i>start()</i> operation exists with correct inheritance implementation. (AP0704)</p> <p><b>Fail:</b> The <i>start()</i> operation does not exist with correct inheritance implementation. (AP0704)</p>		Inheritance is implementation-specific. The tester may find that this requirement has been satisfied in one file, at the super class level. Or, the tester may find the sourcecode in multiple files, at the subclass level.	
6. Examine the sourcecode and verify that the <i>start()</i> operation continues to be processed, if the application is in a stopped state.	<p><b>Pass:</b> The <i>start()</i> operation must be allowed to process, if the application is in a stopped state. (AP0704)</p> <p><b>Fail:</b> The <i>start()</i> operation is not allowed to process, if the application is in a stopped state. (AP0704)</p>			
<b>End of Test</b>				

Test Recording Log – APP_TC_034			
SAD File:			
Step 1 SPD Files	Step 2 sca_compliant SPD Files	Step 3 executable local file name	Step 4 Source Code Files

**Test Summary APP\_TC\_034**

Once testing is complete for every component of the application 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):**

AP0704 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.34APP\_TC\_035 – ResourceFactory::createResource raises CreateResourceFailure

**Test Case Number:** APP\_TC\_035

CF::Resource createResource

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0108	The error number shall indicate a CF ErrorNumberType value.
AP0115	The <i>createResource</i> operation shall raise the <i>CreateResourceFailure</i> exception when it cannot create the resource.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-18 to 3-19, Section 3.1.3.1.7.3.3 and Section 3.1.3.1.7.5.1.5 Page 3-94, Section 3.1.3.6.13
SCA Appendix D: Domain Profile	Version 2.2.2 FINAL / 05-15-2006	Pages D-4 to D-13, Section D-2; D-33 to D-46, and Section D-6
SCA Appendix C: Core Framework IDL	Version 2.2.2 FINAL / 05-15-2006	Page C-3, Section C-1

### Test Objective

This test case verifies AP0108 and AP0115. The objective of this test is to verify that the application's resource factory create resource operation will raise the proper exception when it cannot create a resource. It will also verify that the exception contains an error number of the type CF::ErrorNumberType.

### Places to Verify

Applications or components that require maintaining client applications using the ResourceFactory interface.

### IDL References

#### Data

```
enum ErrorNumberType {  
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,  
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
```

---

CF\_EINVAL, CF\_EIO, CF\_EISDIR, CF\_EMFILE, CF\_EMLINK, CF\_MSGSIZE, CF\_ENAMETOOLONG, CF\_ENFILE, CF\_ENODEV, CF\_ENOENT, CF\_ENOEXEC, CF\_ENOLCK, CF\_ENOMEM, CF\_ENOSPC, CF\_ENOSYS, CF\_ENOTDIR, CF\_ENOTEMPTY, CF\_ENOTSUP, CF\_ENOTTY, CF\_ENXIO, CF\_EPERM, CF\_EPIPE, CF\_ERANGE, CF\_EROFS, CF\_ESPIPE, CF\_ESRCH, CF\_ETIMEDOUT, CF\_EXDEV };

### Exceptions

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

### Operations

```
CF::Resource createResource ( in string resourceId,  
                             in CF::Properties qualifiers )  
    raises (CF::ResourceFactory::CreateResourceFailure);
```

### Preconditions

- The application source code files are available.

### Test Description

- A. Locate all the resource factories implemented by the application. (AP0108, AP0115)
  1. N/A: No ResourceFactory is implemented
- For Each resource factory found perform the following;
- B. Verify that the *CreateResourceFailure* exception is raised when the *createResource* operation cannot create the resource (AP0115).
  1. **Pass:** The *CreateResourceFailure* exception is raised when the *createResource* operation cannot create the resource.
  2. **Fail:** No *CreateResourceFailure* exception is raised.
- C. Verify that the *errorCode* is a CF *ErrorNumberType* value (AP0108)
  1. **Pass:** The *errorCode* is a CF *ErrorNumberType* value
  2. **Fail:** The *errorCode* 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.

APP_TC_035				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>NOTE:</b> The software developer can be of assistance in locating the source code for the resource factory(s). A search engine such as <code>grep</code> or a development tool such as Visual C++ can be used to search all the source code to find the operation. Do not depend on Microsoft explorer's search operation, as it can be demonstrated that it will not find a string within a file.				
<b>A. Locate all the resource factories implemented by the application.</b>				
1. Locate the source code of the resource factories.	<b>N/A:</b> No resource factory is implemented by the application. (AP0115, AP0108)		An application is not required to use resource factories to obtain, create, or tear down resources.	
<b>For each resource factory perform the following;</b>				
<b>B. Verify that the <code>CreateResourceFailure</code> exception is generated when the <i>qualifier</i> is invalid( AP0115).</b>				
2. Locate the <code>createResource</code> operation in the source code.	The <code>createResource</code> operation is located.  Record the name of the source code location.		The operation should be similar to this:  <code>CF::Resource createResource (in string resourceId, in CF::Properties qualifiers) raises (CF::ResourceFactory::CreateResourceFailure)</code>	
3. Verify that when the <code>createResource()</code> operation fails to create a resource, a <code>CreateResourceFailure</code> exception is raised.	<b>Pass:</b> The <code>CreateResourceFailure</code> exception is raised. (AP0115)  <b>Fail:</b> The <code>CreateResourceFailure</code> exception is not raised. (AP0115)			
<b>C. Verify that the <code>errorCode</code> is a <code>CF ErrorNumberType</code> value (AP0108)</b>				

APP_TC_035				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Verify that the error code associated with the exception <i>CreateResourceFailure</i> is a CF ErrorNumberType value.	<b>Pass:</b> The exception includes a CF ErrorNumberType value. (AP0108) <b>Fail:</b> The exception does not include a CF ErrorNumberType value. (AP0108)		The CF ErrorNumberType values are in the CF IDL. And may be found in the IDL section above.	
End of Test				

Test Recording Log – APP_TC_035			
Step 2 (locate <i>createResource</i> code)	Step 3 Raises exception	Steps 4 ErrorNumber Type	Notes

## Test Summary APP\_TC\_035

Once testing is complete for every component of the application 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):**

AP0108 \_\_\_\_\_

AP0115 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.35APP\_TC\_036 – Domain Profile files

### Test Case Number: AP\_TC\_036

APPLICATION Domain Profile

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0613	An application, each application component, and each device manager shall be accompanied by the appropriate Domain Profile files per section 3.1.3.5.
C145	The software profile for an application consists of one SAD file that references (directly or indirectly) one or more SPD, SCD, and properties (PRF) files.
C148	The softpkg id uniquely identifies the package and is a DCE UUID. The DCE UUID is as defined by the DCE UUID standard (adopted by CORBA). The DCE UUID format starts with the characters "DCE:" and is followed by the printable form of the UUID, a colon, a decimal minor version number, for example: "DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1". The decimal minor version number is optional.
C149	All files referenced by a Software Package are located in the same directory as the SPD file or a directory that is relative to the directory where the SPD file is located.
C150	The set of properties to be used for a Software Package come from the union of these properties sources using the following precedence order: 1. SPD Implementation Properties 2. SPD level properties 3. SCD properties Any duplicate properties having the same ID are ignored.
C151	Duplicated properties must be the same property type, only the value can be over-ridden.
C152	The implementation properties are only used for the initial configuration and creation of a component by the CF ApplicationFactory and cannot be referenced by a SAD component instantiation, component properties or resource factory properties element.
C153	The property file element is used to indicate the local filename of the Property Descriptor file associated with the Software Package.
C154	When the name attribute is a simple name, the file exists in the same directory as the SPD file. A relative directory indication begins either with "../" meaning parent directory and "/" means current directory in the name attribute. Multiple "../" and directory names can follow the initial "../" in the name attribute.
C155	The SCD file is optional, since some SCA components are non-CORBA components, like digital signal processor (DSP) "c" code (see section on software component descriptor file, section D.5).
C156	The implementation element is intended to allow multiple component templates to be delivered to the system in one Software Package. Each implementation element is intended to allow the same component to support different types of processors, operating systems, etc.
C157	The implementation element's id attribute uniquely identifies a specific implementation of the component and is a DCE UUID value, as stated in section D.2.1.

SCA v2.2.2 Tag	SCA v2.2.2 Text
C158	The stack size and priority are options parameters used by the CF ExecutableDevice execute() operation.
C159	Data types for the values of these options are unsigned long.
C160	The entrypoint element provides the means for providing the name of the entry point of the component being delivered. The valid values for the type attribute are: "Executable", "KernelModule", "SharedLibrary", and "Driver."
C161	The softpkgref element (see Figure D-7) refers to a softpkgelement contained in another Software Package Descriptor file and indicates a file-load dependency on that file. The other file is referenced by the localfile element.
C162	The propertyref element is used to indicate a uniquerefid attribute that references a simple allocation property, defined in the package, and a property value attribute used by the domain Management function to perform the dependency check. This refid is a DCE UUID, as specified in section D.2.1.
C163	The usesdevice element describes any "uses" relationships this component has with a device in the system. The propertyref element references allocation properties, which indicate the CF Device to be used, and/or the capacity needed from the CF Device to be used.
C164	The id attribute for a simple property that is an allocation type is a DCE UUID value, as specified in section D.2.1
C165	Only simple elements can be used as execparamtypes.
C166	At a minimum, the component interface has to be a CF Resource, CF ResourceFactory, or CF Device interface.
C167	The softwareassembly element's id attribute is a DCE UUID, as specified in section D.2.1, which uniquely identifies the assembly.
C168	The componentfiles element is used to indicate that an assembly is made up of 1..n component files. The componentfile element contains a reference to a local file, which is a Software Package Descriptor file.
C169	The componentfileref element is used to reference a particular Software PackageDescriptor file. The componentfileref element's refid attribute corresponds to the componentfile element's id attribute.
C170	The componentinstantiation's id attribute is a DCE UUID that uniquely identifies the component. The id is a DCE UUID value as specified in section D.2.1.
C171	The optional findcomponent element should be specified except when there is no CORBA object reference for the component instance (e.g., DSP code).

SCA v2.2.2 Tag	SCA v2.2.2 Text
C172	The optional findcomponent element (see Figure D-25) is used to obtain the CORBA object reference for the component instance. The two sources for obtaining a CORBA object reference are: 1. The componentresourcefactoryref element, which refers to a particular CF ResourceFactory componentinstantiation element found in the SAD, which is used to obtain a CF Resource instance for this componentinstantiation element. The refid attribute refers to a unique componentinstantiationid attribute. The componentresourcefactoryref element contains an optional resourcefactoryproperties element (see Figure D-26), which specifies the properties "qualifiers", for the CF ResourceFactory create call." 2. The CORBA Naming Service, which is used to find the component's CORBA object reference. The name specified in the namingservice element is a partial name that is used by the CF ApplicationFactory to form the complete context name.
C173	The usesidentifier element identifies which "uses port" on the component is to participate in the connection relationship. This identifier will correspond with an id for one of the component ports specified in the Software Component Descriptor.
C174	The devicethatloadedthiscomponentref element refers to a specific component found in the assembly, which is used to obtain the logical CF Device that was used to load the referenced component from the CF ApplicationFactory.
C175	The findby element by itself is used when the object reference is not a CF Resource type.
C176	The supportedidentifier element identifies which supported interface on the component is to participate in the connection relationship. This identifier will correspond with the repid attribute of one of the component's supportsinterface elements, specified in the Software Component Descriptor.

## References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Pages 3-95, Section 3.2.1.3; Page 3-90, Section 3.1.3.5
SCA AppendixD: Domain Profile	Version 2.2.2 05-15-2006	All
Spectra SDR Power Tools-User Guide	Version 2.0	All
Spectra XML Validation Capability	01-21-2009, By PrismTech	All

## Test Objective

This test case verifies AP0613. The objective of this test is to verify that the application is described by the XML Domain Profile files properly. It will also build lists for use in other test cases. PrismTech's Spectra SDR Power Tools will be used for this test.

## Places to Verify

application Domain Profile Files

## IDL References

None.

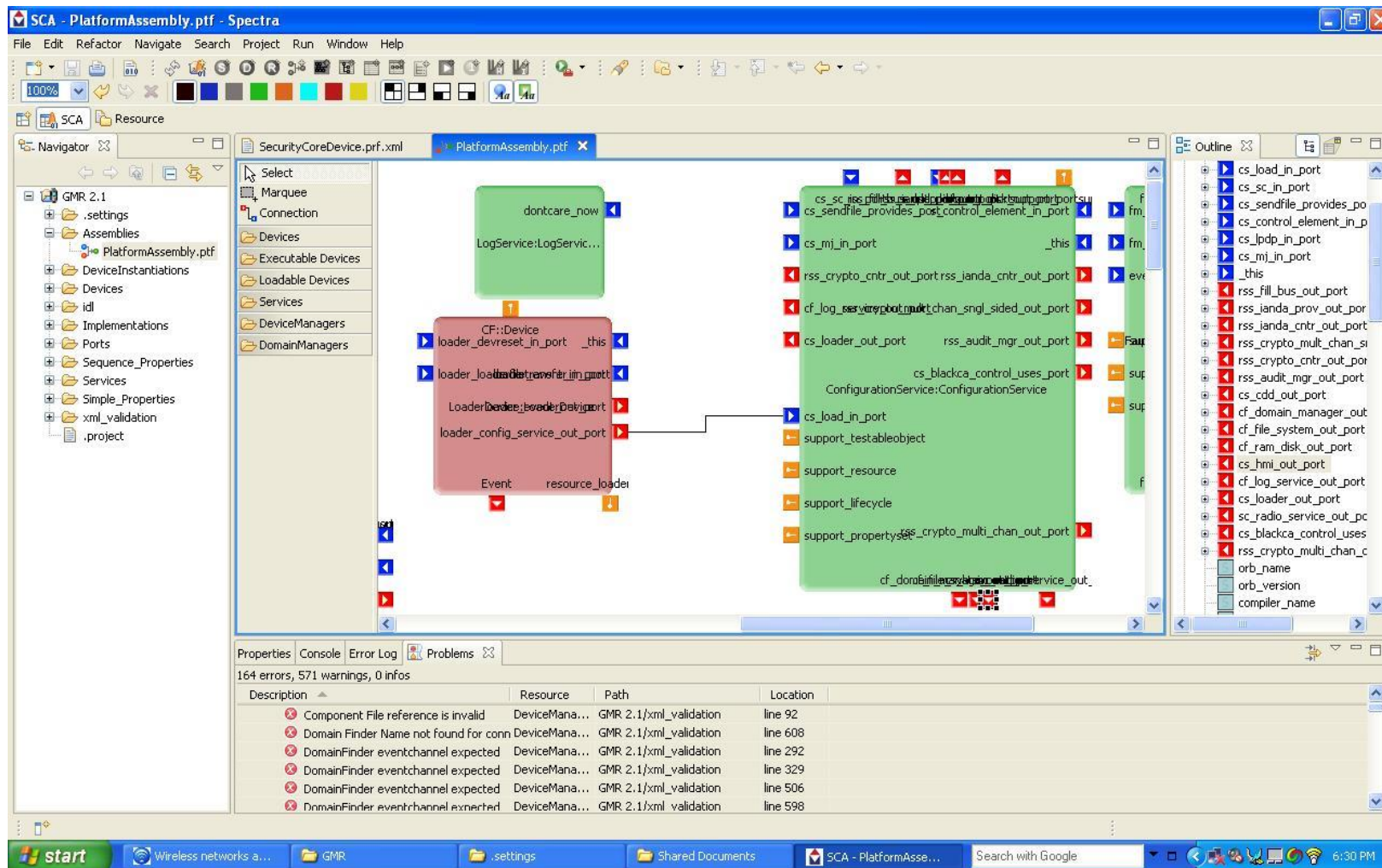
## Preconditions

- The Domain Profile files for the application are available.
- All application files provided by the developer are available.
- The Spectra tool from PrismTech is available.

## Test Description

- A. Set up a workspace for the application and import the developer's files.
- B. Build an SCA Platform Project.
- C. Build the device, service and port lists.
- D. Record and verify all errors and warnings.(AP0613, C145 to C176)
  1. **Pass:** SCA errors are not found.
  2. **Fail:** SCA errors are found.
- E. Verify that the appropriate Domain Profile files are present. (AP0613)
  1. **Pass:** No XML files are missing.
  2. **Fail:** XML files are missing.



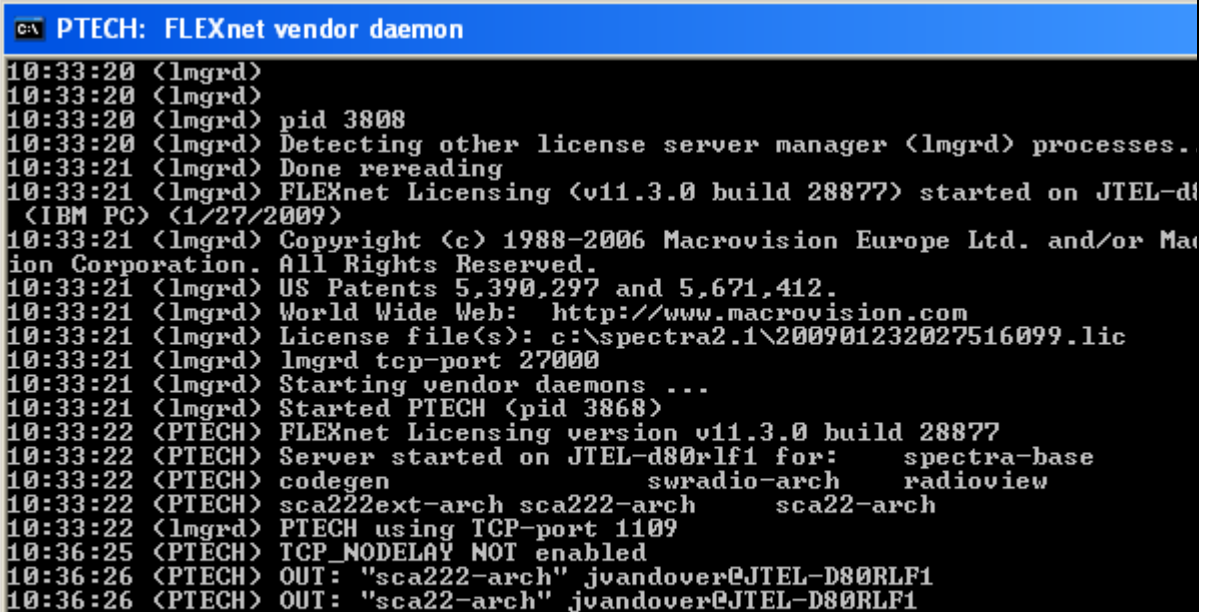


## SPECTRA DESKTOP

## 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.

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Set up a workspace for the APPLICATION and import the developer's files.</b>				
1. Create a workspace directory in your My Documents.	A workspace directory is created.		A workspace is a location where Eclipse and Spectra keep all the files for your projects.	
2. Copy the application XML and IDL files into the Workspace.	Files are successfully copied.		It is recommended that these be put in a directory of their own under the workspace directory.	
3. Start the license server if it does not start up on login. It is named lic_server.bat. It can be found in the window programlist.	A command window opens and the license server starts.	 <pre> CA PTECH: FLEXnet vendor daemon 10:33:20 &lt;lmgrd&gt; 10:33:20 &lt;lmgrd&gt; 10:33:20 &lt;lmgrd&gt; pid 3808 10:33:20 &lt;lmgrd&gt; Detecting other license server manager &lt;lmgrd&gt; processes. 10:33:21 &lt;lmgrd&gt; Done rereading 10:33:21 &lt;lmgrd&gt; FLEXnet Licensing (v11.3.0 build 28877) started on JTEL-d80rlf1 (IBM PC) (1/27/2009) 10:33:21 &lt;lmgrd&gt; Copyright (c) 1988-2006 Macrovision Europe Ltd. and/or Macrovision Corporation. All Rights Reserved. 10:33:21 &lt;lmgrd&gt; US Patents 5,390,297 and 5,671,412. 10:33:21 &lt;lmgrd&gt; World Wide Web: http://www.macrovision.com 10:33:21 &lt;lmgrd&gt; License file(s): c:\spectra2.1\200901232027516099.lic 10:33:21 &lt;lmgrd&gt; lmgrd tcp-port 27000 10:33:21 &lt;lmgrd&gt; Starting vendor daemons ... 10:33:21 &lt;lmgrd&gt; Started PTECH (pid 3868) 10:33:22 &lt;PTECH&gt; FLEXnet Licensing version v11.3.0 build 28877 10:33:22 &lt;PTECH&gt; Server started on JTEL-d80rlf1 for: spectra-base 10:33:22 &lt;PTECH&gt; codegen swradio-arch radioview 10:33:22 &lt;PTECH&gt; sca22ext-arch sca22-arch sca22-arch 10:33:22 &lt;lmgrd&gt; PTECH using TCP-port 1109 10:36:25 &lt;PTECH&gt; TCP_MODELAY NOT enabled 10:36:26 &lt;PTECH&gt; OUT: "sca22-arch" jvandover@JTEL-D80RLF1 10:36:26 &lt;PTECH&gt; OUT: "sca22-arch" jvandover@JTEL-D80RLF1 </pre>		

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
4. Start Spectra.	Spectra starts and a window to select the workspace opens.			
5. Browse to the workspace created in step 1 and select it.	Spectra GUI starts.			
6. Open window-> preferences	The preferences window opens.		We need to insure that the license is setup.	
7. Select Spectra Power Tools	The window changes to display the license selections			
8. Clear the local license field.				
9. Enter the Address of the license server as <i>port@machine</i> then press the apply button	A window pops up showing the license options.		In the picture for step 3, the port is 27000, and the machine is JTEL-d80r1. The FLEXnet licensing line has the machine. Six lines below it is the tcp-port line which has the port FLEXnet is using.	
10. Select ok to close the license and preferences windows.				
<b>B. Buildan SCA Platform Project.</b>				
11. Select <b>File -&gt; New -&gt; SCA Project.</b>	A New SCA Project Window will open		Do <b>not</b> select SCA PlatformProject. An SCA Platformproject is an OE, not an application.	
12. Select <b>SCA v2.2 or SCA v2.2.2</b>				
13. Select <b>Create an SCA Project from Domain Profile xml</b> , then next.	The window will change to a Name Project window			

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
14. Name the project and select next.	The window will change to a Select DMD File window			
15. Browse to the SAD file, select it.	The SAD file will be listed in the window.		There may be several SAD files. Determine which one to use with the aid of the developer's engineer.	
16. Browse to the lowest level directory that contains all the IDL files and select it	The absolute name of the directory selected will appear in the window.			
17. Check the box for search all subfolders, then select finish.	The XML and IDL files will be parsed and the project created.			
<b>C. Build the resource, service and port lists.</b>				
18. Select <b>open Assembly</b> and select <b>PlatformAssembly.sca</b> to bring up the picture.	A picture of the devices, services, and ports will be produced.		This picture can be cleaned up. It is useful for determining the connections between the various components.	

<p>19. In the Outline window expand each resource and service.</p>				
--	--	--	--	--

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
20. Record the resources listed.	A list of resources will be recorded.		resources are a green .	
21. Record the ports listed by name and type (provides or uses).	A list of ports will be presented.		Red ports are output (uses) ports, blue ports are input (consume) ports.	
22. Record the services listed.	A list of services will be presented.		Services are green.	
<b>D. Record all errors and warnings.</b>				
23. Review the problems log and determine if the problems are SCA related.	<b>Pass:</b> None of the problems are SCA related. (AP0613) <b>Fail:</b> One or more problems are SCA related. (AP0613)		Double clicking on a problem in the Spectra log will open the file and take you to the line that has the problem.	
24. Find and double click the Problems Log file.	The ProblemLog file should open as an Excel spreadsheet.		Problems Log is saved based on the SAD file. The Problems Log files will be found in the directory workspace\SAD-name\xml_validation\ in the file RELogyyyy-m-dd-tttt. The file suffix is "cvs".  Where workspace is the name of the Spectra workspace SAD-name is the name of the SAD file (mentioned in step 15) yyyy is the year (always four digits) m is the month (one or two digits) dd is the day (one or two digits) tttt is the time.	
25. Adjust the column widths to improve readability. Then save the result as an xls file	The file will be saved. And much more readable			
<b>For all criteria steps below, success is based on not finding the issue mentioned in the problems log.</b>				

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
26. Within the problems log verify that there is no mention that the SAD file has failed to reference one or more SPD, SCD and PRF files	<b>Pass:</b> The log makes no mention of this. (C145)  <b>Fail:</b> The log makes mention of this. (C145)		Failure of C145 results in failure of the AP0613 requirement.  The software profile for an application consists of one SAD file that references (directly or indirectly) one or more SPD, SCD, and properties (PRF) files.	
27. During the SPD XML validation, search problems log file for the string, "Invalid softpkg id", error#17. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C148)  <b>Fail:</b> An instance of the string was found. (C148)		Failure of C148 results in failure of the AP0613 requirement.  The softpkgid uniquely identifies the package and is a DCE UUID. The DCE UUID is as defined by the DCE UUID standard (adopted by CORBA). The DCE UUID format starts with the characters "DCE:" and is followed by the printable form of the UUID, a colon, a decimal minor version number, for example: "DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1". The decimal minor version number is optional.	
28. During the SCD XML validation, search problems log file for the string, "Referenced file ??? does not exist on the file system", error#3. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C149)  <b>Fail:</b> An instance of the string was found. (C149)		Failure of C149 results in failure of the AP0613 requirement.  All files referenced by a Software Package are located in the same directory as the SPD file or a directory that is relative to the directory where the SPD file is located.  "???" represents the name of missing referenced file.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
29. During the SPD XML validation, search problems log file for the string, "The SPD Implementation property with same ID is not the same definition as specified in SPD or SCD level.", error #28. A search for a partial string may be sufficient to exclude this message.	<p><b>Pass:</b> No instance of the string was found. (C150)</p> <p><b>Fail:</b> An instance of the string was found. (C150)</p>		<p>Failure of C150 results in failure of the AP0613 requirement.</p> <p>The set of properties to be used for a Software Package come from the union of these properties sources using the following precedence order:</p> <ol style="list-style-type: none"> <li>1. SPD Implementation Properties</li> <li>2. SPD level properties</li> <li>3. SCD properties</li> </ol> <p>Any duplicate properties having the same ID are ignored.</p>	
30. During the SPD XML validation, search problems log file for the string, "The SPD property with same ID is not the same definition as specified in SCD: ???", error #29. A search for a partial string may be sufficient to exclude this message.	<p><b>Pass:</b> No instance of the string was found. (C150)</p> <p><b>Fail:</b> An instance of the string was found. (C150)</p>		<p>Failure of C150 results in failure of the AP0613 requirement.</p> <p>"???" represents the name of the related SCD.</p>	
31. Within the problems log verify that there is no mention that duplicate properties had a type mismatch.	<p><b>Pass:</b> The log makes no mention of this. (C151)</p> <p><b>Fail:</b> The log makes mention of this. (C151)</p>		<p>Failure of C151 results in failure of the AP0613 requirement.</p> <p>Duplicated properties must be the same property type, only the value can be over-ridden.</p>	



AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
32. Within the problems log verify that there is no mention that the SPD's implementation properties were sought by other property processes for other Domain Profile files.	<p><b>Pass:</b> The log makes no mention of this. (C152)</p> <p><b>Fail:</b> The log makes mention of this. (C152)</p>		<p>Failure of C152 results in failure of the AP0613 requirement.</p> <p>The implementation properties are only used for the initial configuration and creation of a component by the CF ApplicationFactory and cannot be referenced by a SAD componentinstantiation, componentproperties or resourcefactoryproperties element.</p>	
33. During the SPD XML validation, search problems log file for the string, "Referenced Property file is not a valid Properties XML file.", error#9. A search for a partial string may be sufficient to exclude this message.	<p><b>Pass:</b> No instance of the string was found. (C153)</p> <p><b>Fail:</b> An instance of the string was found. (C153)</p>		<p>Failure of C153 results in failure of the AP0613 requirement.</p> <p>The propertyfile element is used to indicate the local filename of the Property Descriptor file associated with the Software Package.</p>	
34. During the SPD XML validation, search problems log file for the string, "Reference file (e.g., <b>SCD, Property, Implementation SPD, Implementation Code, Implementation Property file</b> ) does not exist on File System.", error#11. A search for a partial string may be sufficient to exclude this message.	<p><b>Pass:</b> No instance of the string was found. (C154)</p> <p><b>Fail:</b> An instance of the string was found. (C154)</p>		<p>Failure of C154 results in failure of the AP0613 requirement.</p> <p>When the name attribute is a simple name, the file exists in the same directory as the SPD file. A relative directory indication begins either with "../" meaning parent directory and "/" means current directory in the name attribute. Multiple "../" and directory names can follow the initial "../" in the name attribute.</p>	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
35. Within the problems log verify that there is no mention that the SCD file is required.	<b>Pass:</b> The log makes no mention of this. (C155)  <b>Fail:</b> The log makes mention of this. (C155)		Failure of C155 results in failure of the AP0613 requirement.  The SCD file is optional, since some SCA components are non-CORBA components, like digital signal processor (DSP) "c" code (see section on software component descriptor file, section D.5).	
36. Within the problems log verify that there is no mention that the application is limited to a single implementation or that the same component is not allowed to support different types of processors, operating systems, etc.	<b>Pass:</b> The log makes no mention of this. (C156)  <b>Fail:</b> The log makes mention of this. (C156)		Failure of C156 results in failure of the AP0613 requirement.  The implementation element is intended to allow multiple component templates to be delivered to the system in one Software Package. Each implementation element is intended to allow the same component to support different types of processors, operating systems, etc.	
37. During the SPD XML validation, search problems log file for the string, "Invalid Implementation ID.???", error #21. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C157)  <b>Fail:</b> An instance of the string was found. (C157)		Failure of C157 results in failure of the AP0613 requirement.  The implementation element's id attribute uniquely identifies a specific implementation of the component and is a DCE UUID value, as stated in section D.2.1.  "???" represents the name of the related SPD.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
38. During the SPD XML validation, search problems log file for the string, "Invalid code stacksize value specified for implementation with id:???", error #13. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C158)  <b>Fail:</b> An instance of the string was found. (C158)		Failure of C158 results in failure of the AP0613 requirement.  The stack size and priority are options parameters used by the CF ExecutableDevice execute() operation.  "???" represents the name of the related SPD.	
39. During the SPD XML validation, search problems log file for the string, "Invalid code priority value specified for implementation with id:???", error #14. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C158)  <b>Fail:</b> An instance of the string was found. (C158)		Failure of C158 results in failure of the AP0613 requirement.  The stack size and priority are options parameters used by the CF ExecutableDevice execute() operation.  "???" represents the name of the related SPD.	
40. Within the problems log verify that there is no mention that the stacksize and priority option parameters are not unsigned long values.	<b>Pass:</b> The log makes no mention of this. (C159)  <b>Fail:</b> The log makes mention of this. (C159)		Failure of C159 results in failure of the AP0613 requirement.  Data types for the values of these options are unsigned long.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
41. During the SPD XML validation, search problems log file for the string, "Invalid code priority value specified for implementation with id:???", error #12. A search for a partial string may be sufficient to exclude this message.	<p><b>Pass:</b> No instance of the string was found. (C160)</p> <p><b>Fail:</b> An instance of the string was found. (C160)</p>		<p>Failure of C160 results in failure of the AP0613 requirement.</p> <p>The entrypoint element provides the means for providing the name of the entry point of the component being delivered. The valid values for the type attribute are: "Executable", "KernelModule", "SharedLibrary", and "Driver."</p> <p>"???" represents the name of the related SPD.</p>	
42. During the SPD XML validation, search problems log file for the string, "Dependency SDP Implementation is not Compatible with specified Implementation", error #3. A search for a partial string may be sufficient to exclude this message.	<p><b>Pass:</b> No instance of the string was found. (C161)</p> <p><b>Fail:</b> An instance of the string was found. (C161)</p>		<p>Failure of C161 results in failure of the AP0613 requirement.</p> <p>The softpkgref element (see Figure D-7) refers to a softpkg element contained in another Software Package Descriptor file and indicates a file-load dependency on that file. The other file is referenced by the localfile element.</p> <p><b>At the time of this writing, May 2009, this error is not supported by the Spectra v1.2.</b></p>	
43. During the SPD XML validation, search problems log file for the string, "Referenced Softpkg is not a softpkg xml file.", error #10. A search for a partial string may be sufficient to exclude this message.	<p><b>Pass:</b> No instance of the string was found. (C161)</p> <p><b>Fail:</b> An instance of the string was found. (C161)</p>		<p>Failure of C161 results in failure of the AP0613 requirement.</p> <p><b>At the time of this writing, May 2009, this error is footnoted that the error message is wrong.</b></p>	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
44. During the SPD XML validation, search problems log file for the string, "The specified Dependency SDP Implementation is not Compatible to implementation", error #27. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C161)  <b>Fail:</b> An instance of the string was found. (C161)		Failure of C161 results in failure of the AP0613 requirement.  <b>At the time of this writing, May 2009, this error is not supported by the Spectra v1.2.</b>	
45. During the SPD XML validation, search problems log file for the string, "The specified Dependency SDP Implementation is not found", error #30. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C161)  <b>Fail:</b> An instance of the string was found. (C161)		Failure of C161 results in failure of the AP0613 requirement.  <b>At the time of this writing, May 2009, this error is not supported by the Spectra v1.2.</b>	
46. Within the problems log verify that there is no mention during the SPD XML validation that one SPD file has a dependency on another SPD file or that the dependency appears to be invalid.	<b>Pass:</b> The log makes no mention of this. (C161)  <b>Fail:</b> The log makes mention of this. (C161)		Failure of C161 results in failure of the AP0613 requirement.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
47. During the SPD XML validation, search problems log file for the string, "Uses device Propertyrefrefid is not a UUID.", error #22. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C162)  <b>Fail:</b> An instance of the string was found. (C162)		Failure of C162 results in failure of the AP0613 requirement.  The propertyref element is used to indicate a unique refid attribute that references a simple allocation property, defined in the package, and a property value attribute used by the domain Management function to perform the dependency check. This refid is a DCE UUID, as specified in section D.2.1.	
48. During the SPD XML validation, search problems log file for the string, "Uses device Propertyref must have a value", error #23. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C162)  <b>Fail:</b> An instance of the string was found. (C162)		Failure of C162 results in failure of the AP0613 requirement.  <b>At the time of this writing, May 2009, this error is footnoted with the following:</b> <ol style="list-style-type: none"> <li>1. Multiple warnings for implementation uses device element property value missing error</li> <li>2. SPD Uses Device element property value missing error not detected.</li> </ol>	
49. During the SPD XML validation, search problems log file for the string, "Implementation dependency Propertyref value is empty.", error #24. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C162)  <b>Fail:</b> An instance of the string was found. (C162)		Failure of C162 results in failure of the AP0613 requirement.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
50. During the SPD XML validation, search problems log file for the string, "Implementation dependency Propertyref refid is not a UUID", error #25. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C162)  <b>Fail:</b> An instance of the string was found. (C162)		Failure of C162 results in failure of the AP0613 requirement.	
51. Within the problems log verify that there is no mention during the SPD XML validation that the usesdevice element failed to describe relationships this component has with devices or the propertyref element failed to reference allocation properties.	<b>Pass:</b> The log makes no mention of this. (C163)  <b>Fail:</b> The log makes mention of this. (C163)		Failure of C163 results in failure of the AP0613 requirement.  The usesdevice element describes any "uses" relationships this component has with a device in the system. The propertyref element references allocation properties, which indicate the CF Device to be used, and/or the capacity needed from the CF Device to be used.	
52. During the PRF XML validation, search problems log file for the string, "Invalid ID for Property:???. Expected a UUID value", error #20. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C164)  <b>Fail:</b> An instance of the string was found. (C164)		Failure of C164 results in failure of the AP0613 requirement.  The id attribute for a simple property that is an allocation type is a DCE UUID value, as specified in section D.2.1  "???" represents the name of the property with the invalid id.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
53. Within the problems log verify that there is no mention during PRF XML validation that elements other than simple elements was used as an execparam type.	<b>Pass:</b> The log makes no mention of this. (C165)  <b>Fail:</b> The log makes mention of this. (C165)		Failure of C165 results in failure of the AP0613 requirement.  Only simple elements can be used as execparamtypes.	
54. Within the problems log verify that there is no mention during SCD XML validation that the component interface was not a CF Resource, CF ResourceFactory, or CF Device interface.	<b>Pass:</b> The log makes no mention of this. (C145)  <b>Fail:</b> The log makes mention of this. (C166)		Failure of C166 results in failure of the AP0613 requirement.  At a minimum, the component interface has to be a CF Resource, CF ResourceFactory, or CF Device interface.	
55. During the SAD XML validation, search problems log file for the string, "Invalid SAD file id.", error #4. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C167)  <b>Fail:</b> An instance of the string was found. (C167)		Failure of C167 results in failure of the AP0613 requirement.  The softwareassembly element's id attribute is a DCE UUID, as specified in section D.2.1, which uniquely identifies the assembly.	
56. During the SAD XML validation, search problems log file for the string, "Referenced SPD file _ does not exist in file system", error #6. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C168)  <b>Fail:</b> An instance of the string was found. (C168)		Failure of C168 results in failure of the AP0613 requirement.  The componentfiles element is used to indicate that an assembly is made up of 1..n component files. The componentfile element contains a reference to a local file, which is a Software Package Descriptor file.	



AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
57. During the SAD XML validation, search problems log file for the string, "Referenced SPD file is not a SPD file type", error #7. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C168)  <b>Fail:</b> An instance of the string was found. (C168)		Failure of C168 results in failure of the AP0613 requirement.  The componentfiles element is used to indicate that an assembly is made up of 1..n component files. The componentfile element contains a reference to a local file, which is a Software Package Descriptor file.  <b>At the time of this writing, May 2009, this error is not supported by the Spectra v1.2.</b>	
58. During the SAD XML validation, search problems log file for the string, "Component File reference "???" not found", error #10. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C169)  <b>Fail:</b> An instance of the string was found. (C169)		Failure of C169 results in failure of the AP0613 requirement.  The componentfileref element is used to reference a particular Software PackageDescriptor file. The componentfileref element's refid attribute corresponds to the componentfile element's id attribute.  "???" represents the name of an unfound file reference.	
59. During the SAD XML validation, search problems log file for the string, "Component Instantiation ID must be a UUID:???", error #11. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C170)  <b>Fail:</b> An instance of the string was found. (C170)		Failure of C170 results in failure of the AP0613 requirement.  The componentinstantiation's id attribute is a DCE UUID that uniquely identifies the component. The id is a DCE UUID value as specified in section D.2.1.  "???" represents a string intended to be a UUID but failing some how.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
60. Within the problems log verify that there is no mention during SAD XML validation that the findcomponent element is missing when there is CORBA object reference for the component instance.	<p><b>Pass:</b> The log makes no mention of this. (C171)</p> <p><b>Fail:</b> The log makes mention of this. (C171)</p>		<p>Failure of C171 results in failure of the AP0613 requirement.</p> <p>The optional findcomponent element should be specified except when there is no CORBA object reference for the component instance (e.g., DSP code).</p>	
61. During the SAD XML validation, search problems log file for the string, "findcomponent not specified for ???", error #16. A search for a partial string may be sufficient to exclude this message.	<p><b>Pass:</b> No instance of the string was found. (C172)</p> <p><b>Fail:</b> An instance of the string was found. (C172)</p>		<p>Failure of C172 results in failure of the AP0613 requirement.</p> <p>The optional findcomponent element (see Figure D-25) is used to obtain the CORBA object reference for the component instance. The two sources for obtaining a CORBA object reference are:</p> <ol style="list-style-type: none"> <li>1. The componentresourcefactoryref element, which refers to a particular CF ResourceFactory componentinstantiation element found in the SAD, which is used to obtain a CF Resource instance for this componentinstantiation element. The refid attribute refers to a unique componentinstantiationid attribute. The componentresourcefactoryref element contains an optional resourcefactoryproperties element (see Figure D-26), which specifies the properties "qualifiers", for the CF ResourceFactory create call."</li> <li>2. The CORBA Naming Service, which is used to find the component's CORBA object reference. The name specified in the namingservice element is a partial name that is used by the CF ApplicationFactory to form the complete context name.</li> </ol> <p>"???" represents a component name.</p> <p><b>Note: The processing does not do check for Resource Factory.</b></p>	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
62. Within the problems log verify that there is no mention during the SAD XML validation that the CORBA object reference could not be obtained because (1) failures in determining the ResourceFactory or (2) failures in determining the CORBA Naming Service.	<b>Pass:</b> The log makes no mention of this. (C172) <b>Fail:</b> The log makes mention of this. (C172)		Failure of C172 results in failure of the AP0613 requirement.	
63. During the SAD XML validation, search problems log file for the string, "The usesidentifier must be a valid reference: ???", error #21. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C173) <b>Fail:</b> An instance of the string was found. (C173)		Failure of C173 results in failure of the AP0613 requirement.  The usesidentifier element identifies which "uses port" on the component is to participate in the connection relationship. This identifier will correspond with an id for one of the component ports specified in the Software Component Descriptor.  "???" represents the invalid usesidentifier id.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
64. During the SAD XML validation, search problems log file for the string, "The devicethatloadedthiscomponentref must be a valid component instantiation reference within SAD", error #25. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C174)  <b>Fail:</b> An instance of the string was found. (C174)		Failure of C174 results in failure of the AP0613 requirement.  The devicethatloadedthiscomponentref element refers to a specific component found in the assembly, which is used to obtain the logical CF Device that was used to load the referenced component from the CF ApplicationFactory.  <b>At the time of this writing, May 2009, this error is not supported by the Spectra v1.2.</b>	
65. Within the problems log verify that there is no mention during the SAD XML validation that element, devicethatloadedthiscomponentref (one word), was invalid."	<b>Pass:</b> The log makes no mention of this. (C174)  <b>Fail:</b> The log makes mention of this. (C174)		Failure of C174 results in failure of the AP0613 requirement.	
66. Within the problems log verify that there is no mention during the SAD XML validation that the findby element is used with other information when the object reference is not a CF Resource type.	<b>Pass:</b> The log makes no mention of this. (C175)  <b>Fail:</b> The log makes mention of this. (C175)		Failure of C175 results in failure of the AP0613 requirement.  The findby element by itself is used when the object reference is not a CF Resource type.	

AP_TC_036				
Steps	Expected Results	Actual Results	Comments	Test Result
67. During the SAD XML validation, search problems log file for the string, "The supportsidentifier must be a valid reference: ???", error #23. A search for a partial string may be sufficient to exclude this message.	<b>Pass:</b> No instance of the string was found. (C176)  <b>Fail:</b> An instance of the string was found. (C176)		Failure of C176 results in failure of the AP0613 requirement.  The supportedidentifier element identifies which supported interface on the component is to participate in the connection relationship. This identifier will correspond with the repid attribute of one of the component's supportsinterface elements, specified in the Software Component Descriptor.  "???" represents the invalid supportsidentifier id.	
<b>E. Verify that the appropriate Domain Profile files are present. (AP0613)</b>				
70. Verify that there are no errors related to missing XML files.	<b>Pass:</b> No XML files are missing. (AP0613)  <b>Fail:</b> XML files are missing. (AP0613)		Spectra will complain if executable files are missing. This is not an SCA violation.	
<b>End of test</b>				

Test Recording Log – AP_TC_036						
Step 20 (Resources)	Step 21 (Ports)	Step 22 (Services)	Step 23 (Error Log File)	Step 24 (Problem Log File)	Step 25 (Errors)	Step 68 (XML files missing)

Test Recording Log – AP_TC_036						
Step 26 (C145 )	Step 27 (C148)	Step 28 (C149)	Step 29 (C150)	Step 30 (C150)	Step 31 (C151)	Step 32 (C152)
Step 33 (C153)	Step 34 (C154)	Step 35 (C155)	Step 36 (C156)	Step 37 (C157)	Step 38 (C158)	Step 39 (C158)
Step 40 (C159 )	Step 41 (C160)	Step 42 (C161)	Step 43 (C161)	Step 44 (C161)	Step 45 (C161)	Step 46 (C161)
Step 47 (C162 )	Step 48 (C162)	Step 49 (C162)	Step 50 (C162)	Step 51 (C163)	Step 52 (C164)	Step 53 (C165)
Step 54 (C166)	Step 55 (C167 )	Step 56 (C168)	Step 57 (C168)	Step 58 (C169)	Step 59 (C170)	Step 60 (C171)
Step 61 (C172)	Step 62 (C172 )	Step 63 (C173)	Step 64 (C174)	Step 65 (C174)	Step 66 (C175)	Step 67 (C176)

## Test Summary AP\_TC\_036

Once testing is complete for every component of the application 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):

AP0613	_____	C162	_____
C145	_____	C163	_____
C148	_____	C164	_____
C149	_____	C165	_____
C150	_____	C166	_____
C151	_____	C167	_____
C152	_____	C168	_____
C153	_____	C169	_____
C154	_____	C170	_____
C155	_____	C171	_____
C156	_____	C172	_____
C157	_____	C173	_____
C158	_____	C174	_____
C159	_____	C175	_____
C160	_____	C176	_____
C161	_____		

### Failed Items (Section/StepNumber):

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

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

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

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



## B.36APP\_TC\_037 – Standard arguments for executable components

**Test Case Number:** APP\_TC\_037

CF::Interfaces

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0612	Each executable component of an application shall accept the standard argv arguments of the POSIX exec family of functions [4].

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-95, Section 3.2.1.3
SCA AppendixD	Version 2.2.2 05-15-2006	Pages D-33 – D37

### Test Objective

This test case verifies AP0612. The objective of this test is to verify that each executable component within the application accepts the standard argv arguments of the POSIX exec family of functions.

### Places to Verify

Verify at the entry point of the application under test.

### IDL Reference

None.

### Preconditions

- The application source code files are available.

## Test Description

For each component: If no component names are specified by the tester, all components found within the SAD will be tested. Start in the SAD; check each SPD and implementation element (found in the SPD XML file).

For each component:

- A. From the SPDs that are compliant, locate each declaration of executable files. For each executable file, perform a search on the associated source code to locate its entry point and verify that its entry point accepts the argument *argv[]* (AP0612). (For some languages the entry point is *main()* but other languages allows you to assign any procedure as its entry point.)
  1. **Pass:** The executable files entry point accepts the argument *argv[]*.
  2. **Fail:** The executable files entry point does not accept an argument of type *argv[]*.

## 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.

APP_TC_037				
Steps	Expected Results	Actual Results	Comments	Test Result
For each component.				
A. From the SPDs that are compliant, locate each declaration of executable files. For each executable file, perform a search on the associated source code to locate its entry point and verify that its entry point accepts the argument <i>argv[]</i> . (AP0612)				
1. Locate and examine the source code files of each resource for its entry point.	<b>Pass:</b> Its entry point exists. (AP0612)  <b>Fail:</b> Its entry point does not exist. (AP0612)		Consult the developer for locations of resource entry points. For some languages the entry point is <code>main()</code> but other languages allow you to assign any procedure as its entry point.	
2. Verify that the entry point accepts the argument <i>argv[]</i> .	<b>Pass:</b> The entry point accepts the argument <i>argv[]</i> (AP0612)  <b>Fail:</b> The entry point does not accept the argument <i>argv[]</i> (AP0612)			
End of Test				

Test Recording Log – APP_TC_037		
Step 1 (Entry point)	Step 2 (argv[])	Notes

**Test Summary APP\_TC\_037**

Once testing is complete for every component of the application 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):**

AP0612\_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.37 APP\_TC\_039 – Requirements for IDL mappings

**Test Case Number:** APP\_TC\_039

Application Interfaces

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0741	All non-standard interfaces shall be defined in Interface Control Documents that are available to other parties without restriction to the extent that interfacing or replacement hardware and software can be developed by other parties without restriction.
AP0742	All SCA APIs shall have their interfaces described in IDL.
AP0743	All non-IDL interfaces shall provide an IDL mapping within the service definition.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-95 (ports) Page 3-96 Service Definitions
SCA, Appendix C: IDL	Version 2.2.2 05-15-2006	IDL
The Common Object Request Broker: Architecture and Specification, Object Management Group, Inc (OMG)	Version 3.0 07-2002	Chapter 3 OMG IDL Pages 3-2; 3-20 –3-26
JTNC SCA Developer's Guide	Revision 1.1 06-18-2002	Section 2.6.1 Page 19 of 78;

### Test Objective

This test case verifies AP0741, AP0742, and AP0743. The objective of this test is to verify that any external interfaces are defined in a document that is available to other parties allowing them to design interfaces to the applications. The test depends upon the usability of the available non-standard interface documentation. The test case emphasis is on the application, so testing will focus on software, rather than the hardware. Therefore, all APIs implemented in a JTR set have their interfaces generated through the use of an IDL. Additionally, this test verifies that APIs generated using methods other than the use of an IDL must be documented in the service definition mapping format. The order of testing the requirements should be to test AP0742 first then AP0741, and lastly AP0743.

## Standard vs Non-standard Interfaces

The criteria of a ‘standard’ interface (and thereby, inferring a ‘non-standard’ interface) is one from a provider who is a recognized Standards organization. For example, interfaces complying with the JTNC Standardized API specifications would be considered standard. The burden of proof is on the developer/vendor to provide the convincing documentation on defining the non-standard interfaces. Keep in mind the fundamental basis, spirit and intention for the requirement (AP0741): that any non-standard, proprietary-type of interface to the application component can be given away without restrictions to someone other than the developer/vendor so that others can port the application in the future.

## Preconditions

- Have a clear understanding/agreement between the developer/vendor and the tester of what a standard and a non-standard interface are. The developer/vendor bears the burden of proof with convincing documentation of the non-standard interface descriptions with quality, quantity and completeness.
- ICD document must be available.
- A list of all APIs and their version must be available.
- The application source code files must be available.
- All files that are used by the application (e.g. Domain Profile, IDL) must be available.

## Places to Verify

Verify in the SCA IDL and the IDL mapping within the service definition.

## IDL References

None

## Test Description

For each component of the application under test:

- A. Identify all the APIs declared in the Domain Profile. (AP0741, AP0742, AP0743)
- B. Verify that all APIs implemented in a JTR set have their interfaces generated through the use of an IDL. (AP0742)
  1. **Fail:** Not all APIs are described and generated through the use of an IDL.
  2. **Pass:** All APIs are described and generated through the use of an IDL.

- 
- C. Find all interface documents for each component's software. The documented descriptions include both API and non-API interfaces. An example of the interface document is the Interface Control Documents which include specifications, diagrams and other interface descriptions. Emphasis is on software, rather than hardware. (AP0741)
1. **Untested:** If unable to locate interface documents.
- D. From the interfaces found in Step C above, identify all non-standard interfaces from the list of interfaces, including which interfaces involve proprietary software. (AP0741)
1. **Untested:** If unable to differentiate between standard and non-standard interfaces.
- E. Using the definitions to differentiate between the standard and non-standard interfaces resulting from the Preconditions section above, verify that all of the non-standard interfaces are reasonably defined in the Interface Control Documents and any other documents. For each of the non-standard interfaces, verify that the interface descriptions contain enough interface information. (AP0741)
1. **Pass:** The quality, quantity and completeness of the non-standard interface descriptions that have been documented and available are defined adequately for another developer to continue the software development for the interfacing or replacement software.
  2. **Fail:** There is less than a minimal set of non-standard interface information that has been documented and available. This completeness level can be somewhat subjective, but minimal means that the scope of coverage and level of detail exists for another developer to continue the software development for the interfacing.
- F. Verify each API that is not generated using methods other than the use of IDL must be documented in the service definition mapping format. (AP0743)
1. **Fail:** Not all non-IDL generated API are documented.
  2. **Pass:** All non-IDL generated API are documented.

*Note: Service Definition is explained in the referenced CORBA Specification.*



## 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.

APP_TC_039				
Steps	Expected Results	Actual Results	Comments	Test Results
<b>A. Identify all the APIs declared in the Domain Profile. (AP0741, AP0742, AP0743)</b>				
1. Locate and open the SAD file for the application.	<b>Pass:</b> The SAD file is located. (AP0741, AP0742, AP0743)  <b>Untested:</b> There is no SAD file. (AP0741, AP0742, AP0743)			
2. Identify and record all Software Package Descriptor (SPD) declarations in the SAD file.	<b>Pass:</b> The SPD declarations are recorded. (AP0741, AP0742, AP0743)  <b>Fail:</b> There are no SPD declarations found. (AP0741, AP0742, AP0743)		<i>Sample declaration:</i>  <componentfile id = "xxxxx"> <localfile name= "xxxxx.spd.xml" />	
3. Identify and record all Software Component Descriptor (SCD) declarations from each of the SPDs declared in Step 2.	<b>Pass:</b> The SCD declarations are recorded. (AP0741, AP0742, AP0743)  <b>Fail:</b> There are no SCD declarations found. (AP0741, AP0742, AP0743)		<i>Sample declaration:</i> <descriptor> <localfile name= "xxxxx.scd.xml" />	
4. Identify and record all the <i>supportsinterface</i> declarations from the each of the SCDs declared in Step 3.	<b>Pass:</b> The <i>supportsinterface</i> declarations are recorded. (AP0741, AP0742, AP0743)  <b>Untested:</b> There are no <i>supportsinterface</i> declarations. (AP0741, AP0742, AP0743)		<i>Sample declaration:</i> <componentfeatures> <supportsinterface repid= "IDL:CF/yyyyy" supportsname="yyyyy" />	
5. Identify all the <i>uses</i> interfaces declarations from the each of the SCDs declared in Step 3.	<b>Pass:</b> The <i>uses</i> interface declarations are recorded. (AP0741, AP0742, AP0743)  <b>Untested:</b> There are no <i>uses</i> interface declarations. (AP0741, AP0742, AP0743)		<i>Sample declaration:</i> <ports> <uses repid="IDL:CF/zzzzz" usesname="zzzzz">	

APP_TC_039				
Steps	Expected Results	Actual Results	Comments	Test Results
6. Identify and record all the <i>provides</i> interfaces declarations from the each of the SCDs declared in Step 3.	<p><b>Pass:</b> The <i>provides</i> interface declarations are recorded. (AP0741, AP0742, AP0743)</p> <p><b>Untested:</b> There are no <i>provides</i> interface declarations. (AP0741, AP0742, AP0743)</p>		<p>Sample declaration:  <code>&lt;provides repid="IDL:nnnnn" providesname="nnnnn"&gt;</code></p>	
<b>B. Verify that all APIs implemented in a JTR set have their interfaces generated through the use of an IDL. (AP0742)</b>				
<p>7. Examine each interface declared in Steps 4 to 6 and verify that each interface is described and generated through the use of an IDL file.</p> <p>Record the IDL generated interfaces and the non-IDL generated interfaces.</p>	<p><b>Pass:</b> Each interface declared is described and generated through the use of an IDL file. Interfaces are recorded. (AP0742)</p> <p><b>Fail:</b> Each interface declared is not described and generated through the use of an IDL file. Interfaces are recorded. (AP0742)</p>		<p><b>interface</b> Aaaaa : Bbbbb {              struct AaaBbbType {                  string CccID;                  unsigned long Ddddd;              };  <b>interface</b> GgggHhhh {              exception UnknownGggg { };          }</p>	
<b>C. Verify that each API that is not generated using methods other than the use of IDL must be documented in the service definition mapping format. (AP0743)</b>				
<p>8. Locate and write down the name of all resource components declared in the SAD file in the Log Sheet.</p> <p>Resources are declared in the <code>&lt;componentfile&gt;</code> element of the SAD file.</p>			<p>Interface declarations are in the SCD files, so review the SAD which points to the SPD which then points to the SCD.          (Note: bolded part of name is vendor-specific example.)</p> <p><b>SAD examination:</b> Locating the componentfile and SPD file declarations in the SAD file.  <code>&lt;componentfile id="BlackComponent_File"&gt;</code></p>	

APP_TC_039				
Steps	Expected Results	Actual Results	Comments	Test Results
			<pre>&lt;localfile name="BlackComponent.spd.xml"/&gt; &lt;/componentfile&gt;  SPD Examination: From the SPD, locate and record the SCD property file declaration. &lt;/propertyfile&gt;&lt;descriptor&gt;&lt;localfile name="BlackComponent.scd.xml"/&gt; &lt;/descriptor&gt;  SCD Examination: From the SCD, locate and record all the interfaces declarations. &lt;interfaces&gt; &lt;interface repid="IDL:CF/Resource:1.0" name="Resource"&gt; interfaces &lt;inherits interface repid="IDL:CF/PortSupplier:1.0"/&gt; ... &lt;/interfaces&gt;</pre>	

APP_TC_039				
Steps	Expected Results	Actual Results	Comments	Test Results
9. Examine all vendor-provided interface documents for software interface descriptions and tag/identify the locations for the next test step	Documents are examined for subsequent test steps.  <b>Untested:</b> If unable to locate interface documents. (AP0741)		Application interfaces focus on software. Hardware interfaces addressed in the Operating System (OE) test case, tentatively identified as OE_TC_163.  Interfaces within and between applications to be considered are: <ol style="list-style-type: none"> <li>1. Base Application interfaces (such as, Port, Port Supplier, Resource, ResourceFactory, LifeCycle, Testable Object, and PropertySet interfaces)</li> <li>2. Framework Control interfaces (such as Device interfaces and Domain interfaces)</li> <li>3. Framework Services File interfaces (such as File, FileSystem and FileManager interfaces)</li> </ol> Note: An example of the interface document would be the Interface Control Documents which would include specifications, diagrams and other interface descriptions.	
<b>D. From the interfaces found in Step C above, identify all non-standard interfaces from the list of interfaces, including which interfaces involve proprietary software. (AP0741)</b>				

APP_TC_039				
Steps	Expected Results	Actual Results	Comments	Test Results
<p>10. List all of the interfaces from Step 9 above that appear to be non-standard interfaces. This list also includes proprietary hardware and software interfaces.</p> <p>If only standard interfaces are discovered, then skip to the End of the Test.</p>	<p><b>Untested:</b> (May or may not find non-standard interfaces.) (AP0741)</p>		<p>Note: The following criteria of a 'standard' interface (and thereby, infers a 'non-standard' interface) is an interface from a provider who is a recognized Standards organization. For example, interfaces complying with the JTNC Standardized API specifications would be considered standard. The burden of proof is on the developer/vendor to provide the convincing documentation on defining the non-standard interfaces.</p>	
<p><b>Using the definitions to differentiate between the standard and non-standard interfaces resulting from the Preconditions section above, verify that all of the non-standard interfaces are reasonably defined in the Interface Control Documents and in any other documents. (AP0741)</b></p>				
<p>11. For each of the non-standard interfaces, verify that the interface descriptions in the Interface Control Documents contain enough details that another party can develop interface code for the hardware and/or software.</p>	<p>This completeness level can be somewhat subjective, but 'enough' means that the scope of coverage and level of detail provides enough specifics for another developer to continue the software development for the interfacing or replacement hardware.</p> <p><b>Pass:</b> The ICD contain information to properly describe the non-standard interfaces. (AP0741)</p> <p><b>Fail:</b> The ICD does not contain "enough" information to properly describe the non-standard interfaces. (AP0741)</p>		<p>Note: Keep in the mind the fundamental basis, spirit and intention for the requirement: that any non-standard, proprietary-type of interface to the application component can be given away without restrictions to someone other than the developer/vendor so that others can port the application in the future. Because of the very nature that the interface is not standard and not industry-wide accepted and known requires that this interface essentially be 'open sourced'.</p>	
<p><b>E. Verify each API that is not generated using methods other than the use of IDL must be documented in the service definition mapping format. (AP0743)</b></p>				

APP_TC_039				
Steps	Expected Results	Actual Results	Comments	Test Results
12. Examine each interface declared in Steps 4 to 6 that is non-IDL generated that it is documented following the service definition mapping format.	<b>Pass:</b> Each non-IDL generated API is documented following the service definition. (AP0743)  <b>Fail:</b> Each non-IDL generated API is not documented following the service definition. (AP0743)		These APIs should be documented similar to the API descriptions in Appendix C of SCA222.	
End of Test				

Test Recording Log – APP_TC_039				
Step 3 (SCD Files)	Steps 4,5,6 (List of APIs)	Step 7 (In IDL format)	Step 7 (Not in IDL format)	Step 8 (Resource Components: SPD and SCD files)
Step 9 (List of interface documents)	Steps 10 (List of non-standard interfaces)	Step 11 (Reasonable details and completeness of non- standard interfaces (Yes/No))	Step 12 (Mapped in Service Definition)	

## Test Summary APP\_TC\_039

Once testing is complete for every component of the application 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):**

AP0741 \_\_\_\_\_

AP0742 \_\_\_\_\_

AP0743 \_\_\_\_\_

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

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

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

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

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



## B.38APP\_TC\_040 – General Rules (Legacy Software)

**Test Case Number:** APP\_TC\_040

Legacy Software Interfaces

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0630	Legacy software shall interface with the Core Framework in accordance with this specification.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-98, Section 3.4.1.2
Support and Rationale Document (SRD) for the SCA	Version 2.2 12-19-2001	Page 3-31

### Test Objective

This test case verifies AP0630. The objective of this test is to verify the SCA compliance of legacy software components of the application that interface with the Core Framework. Even though the term, “legacy”, is not defined in the SCA document, the generally accepted definition is software components of the radio that were produced before the application’s development for SCA compliancy was started. It is not a requirement that a legacy component be developed in a Higher Order Language but the legacy software is required to be SCA compliant when interfacing with the Core Framework. The legacy software must use the SCA compliant CF Interfaces for Base Application Interfaces and Devices defined in Section 3.1.3 of the SCA v2.2.2 document and follow the guidance for legacy software provided in the Support and Rationale Document (SRD) for the SCA (v2.2)

### Places to Verify

Any place that legacy software is used must be checked to ensure it is properly interfaced to the Core Framework in the application under test.

### IDL References

None.

## Preconditions

- The source code of the application and the legacy software is available.
- SCA related software design documentation should be available: Software Design Document (SDD), Software Version Description (SVD), Software Requirements Specification (SRS) or any other documentation that describes the legacy code interface with respect to the other parts of the application's SCA architecture.

## Test Description

- A. From the application design documentation that is provided by the developer, identify any legacy software including fpga or dsp images for an application(AP0630).
  1. **N/A:** The legacy software exists in the application but is not currently being used (i.e. dead code). It could be in violation of other JTNC requirements but does not specifically fail SCA Requirements
- B. Determine where the legacy software interfaces with the Core Framework, i.e. the SCA component that it is part of.. (AP0630)
  1. **Fail:** Cannot find its Core Framework interface.
- C. Verify that the component, that the legacy software is part of, implements the CF Interfaces for a Resource as defined by the SCA. It must then implement the CF Interfaces specified for the mandatory Base Application Interfaces: *LifeCycle*, *TestableObject*, *PropertySet*, *PortSupplier*. It may in some cases follow the CF Interfaces for a *ResourceFactory*, but this interface is optional(AP0630)
  1. **Fail:** The legacy software is being used in the application but cannot be identified with a part of the application's SCA architecture. The component does not follow the CF Interfaces specified for the Base Application Interfaces as described in the SCA v2.2.2, Section 3.1.3.1
  2. **Pass:** The legacy software of the application identified, documented and follow the CF Interfaces defined for either a *Resource*; therefore interfacing with Core Framework according to the SCA

## 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.

APP_TC_040				
Steps	Expected Results	Actual Results	Comments	Test Results
<b>A. From the application design documentation that is provided by the developer, identify any legacy software including fpga or dsp images for an application. (AP0630)</b>				
1. Identify the part of the SCA Architecture that the legacy component is identified or integrated with. The component should be identified as part of the radio's software application(OE is a separate test). It should be specified even further as being identified or interfaced with a <i>Resource, Device</i> or other architectural component of the SCA.	<p>The legacy component should be identified as interfaced or integrated with a part of the SCA Architecture. (AP0630)</p> <p>N/A: The legacy software exists in the application but is not currently being used (i.e. dead code). The test can stop here.</p>			
<b>B. Determine where the legacy software interfaces with the Core Framework, i.e. the SCA component that it is part of.. (AP0630)</b>				
2. Identify where the legacy software interfaces with the Core Framework, i.e. the SCA component that it is part of.. (AP0630)	<p>Legacy software interfaces are found. (AP0630)</p> <p><b>Fail:</b> If there is no legacy software in the application, then the test can stop here.</p>			
<b>C. Verify that the component, that the legacy software is part of, implements the CF Interfaces for a Resource as defined by the SCA. It must then implement the CF Interfaces specified for the mandatory Base Application Interfaces: LifeCycle, TestableObject, PropertySet, PortSupplier. (AP0630) It may in some cases follow the CF Interfaces for a Resource Factory, but this interface is optional.</b>				

APP_TC_040				
Steps	Expected Results	Actual Results	Comments	Test Results
3. Verify that the Resource the legacy software is associated with follows the CF Interface specifications for a <i>LifeCycle</i> , <i>PropertySet</i> , <i>TestableObject</i> , <i>PortSupplier</i> described in the SCA v2.2.2, section 3.1.3.1. (AP0630)	The Resource follows the design specifications for Resources defined in the Base Application Interfaces Section of the SCA v2.2.2, section 3.1.3.1. (AP0630)		Components to consider for this testing are GPP (Resource, ResourceFactory), and FPGA and DSP images.	
Optional Step 4. : Only include this step if a Resource Factory is implemented. If the component is a <i>ResourceFactory</i> , verify that it follows the CF Interface specifications for a <i>ResourceFactory</i> as described in the SCA v2.2.2, section 3.1.31. (AP0630)	The Resource follows the design specifications for a <i>ResourceFactory</i> defined in the Base Application Interfaces Section of the SCA v2.2.2, section 3.1.3.1. (AP0630)			
<b>End of Test</b>				

Test Recording Log – APP_TC_040			
Step 1	Step 2	Step 3	Step 4
Design document name and page.	Source code file name for legacy software interface	Does the legacy software adhere to the specifications for a Base Application Interface of the SCA v2.2.2?	Optional Step: Only include this step if a <i>ResourceFactory</i> is implemented.  Is the legacy software a Resource Factory and conforms to the SCA?

## Test Summary APP\_TC\_040

Once testing is complete for every component of the application 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):**

AP0630 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.39APP\_TC\_041 – OS Services (File access)

**Test Case Number:** APP\_TC\_041

Applications::CF File Interfaces

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0604	Applications shall perform file access through the CF File interfaces.

### References

Document Name	Version/Date	Location
SCA	Version 2.2.2 05-15-2006	Page 3-95, Section 3.2.1.1

### Test Objective

This test case verifies AP0604. The objective of this test is to verify that the application, when using file access operations, uses the Core Framework (CF) File interfaces. This verification is accomplished through the examination of the application source code files.

### Places to Verify

Applications that implement file access operations.

### IDL References

#### Operations

```
CF::File create (  
    in string fileName )  
    raises (CF::InvalidFileName, CF::FileException);  
CF::File open (  
    in string fileName,  
    in boolean read_Only )  
    raises (CF::InvalidFileName, CF::FileException);
```

```
void read (  
    out CF::OctetSequence data,  
    in unsigned long length )  
    raises (CF::File::IOException);  
void write (  
    in CF::OctetSequence data )  
    raises (CF::File::IOException);
```

## Preconditions

- The application source code files are available.

## Test Description

For each component: If no component names are specified by the user, all components found within the SAD will be tested. Start in the SAD; check each SPD and implementation element.

For each component:

- A. For each executable component, verify from the source code files that the application is using only file access services through the CF File interfaces. (AP0604)
  1. Identify any file accesses made within the code.
    - a. **N/A:** The application is not using any file access services.
    - b. **Pass:** The application is using file access services through the CF File interfaces.
    - c. **Fail:** The application is using file access services but not through the CF File interfaces.



## 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.

APP_TC_041				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Verify from the source code files that the application is using only file access services through the <i>CF File</i> interfaces. (AP0604)</b>				
1. Locate and record the corresponding source code files for the executable source files.	The corresponding source code files will be recorded.		Record source code file name in the test log.	
2. Examine each source code (Step 5) and locate all file access operations and verify that <i>CF File</i> interfaces are used for file access.	<p><b>N/A:</b> The application is not using any file access services. (AP0604)</p> <p><b>Pass:</b> The application is using file access services through the CF File interfaces. (AP0604)</p> <p><b>Fail:</b> The application is using file access services but not through the CF File interfaces. (AP0604)</p>		Record CF::File accesses in the test log. These file access operations, without the <b>CF::File</b> , are direct POSIX calls and will fail the test: <i>open()</i> , <i>read()</i> , <i>fread()</i> , <i>write()</i> , <i>fwrite()</i> , <i>close()</i> verify that these operations are only being used with respect to an SCA FileSystem. Also record other (non-compliant) file accesses in the test log. The application should not be using C++ native file access -ofstream, ifstream, fstream.	
<b>End of Test</b>				

Test Recording Log – APP_TC_041		
Step 1 Source Code File	Step 2 <i>CF::File accesses</i>	Step 2 Non <i>CF::File</i> accesses

## Test Summary APP\_TC\_041

Once testing is complete for every component 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):**

AP0604 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.40 APP\_TC\_042 – OS Services (SIGQUIT)

**Test Case Number:** APP\_TC\_042

Applications: *SIGQUIT* Signal

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0606	All application processes shall have a handler registered for the POSIX-defined <i>SIGQUIT</i> signal.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-95, Section 3.2.1.1
SCA AppendixD, Domain Profile	Version 2.2.2 05-15-2006	Page D-36

### Test Objectives

This test case verifies AP0606. The objective of this test is to verify that all application processes shall have a handler registered for the POSIX-defined *SIGQUIT* signal. *SIGQUIT* is an OS defined function that controls and creates a graceful termination of the application..

### Places to Verify

Applications that implement a handler registered for the POSIX-defined *SIGQUIT*.

### IDL References

None

### Preconditions

- The application source code files are available.

### Test Description

For each application and application component:

- A. Identify the source code file(s) that implements the POSIX-defined *SIGQUIT* signal handler. (AP0606)

1. **Pass:** An implementation is identified that uses the POSIX-defined *SIGQUIT* signal handler.
  2. **Fail:** There are no implementations that use the POSIX-defined *SIGQUIT* signal handler.
- B. Verify that the source code file(s) have a handler registered for the POSIX-defined *SIGQUIT* signal. (AP0606)
- a. **Pass:** An implementation of a handler registered for the POSIX-defined *SIGQUIT* signal exists.
  - b. **Fail:** An implementation for a handler registered for the POSIX-defined *SIGQUIT* signal 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.

APP_TC_042				
Steps	Expected Results	Actual Results	Comments	Test Result
For each application and application component:				
A. Identify the source code file(s) that implements the POSIX-defined <i>SIGQUIT</i> signal handler. (AP0606)				
1. Identify the source code file(s) implementing the <i>SIGQUIT</i> signal handler. (AP0606)	<b>Pass:</b> An implementation is identified that uses the POSIX-defined <i>SIGQUIT</i> signal handler. (AP0606)  <b>Fail:</b> There are implementations that use the POSIX-defined <i>SIGQUIT</i> signal handler. (AP0606)		To locate the source code file(s), the software engineer can be of assistance.	
B. Verify that the source code file(s) implements a handler registered for the POSIX-defined <i>SIGQUIT</i> signal. (AP0606)				

APP_TC_042				
Steps	Expected Results	Actual Results	Comments	Test Result
2. Verify that the source code implements a handler registered for the <i>SIGQUIT</i> signal.	<b>Pass:</b> An implementation of a handler registered for the POSIX-defined <i>SIGQUIT</i> signal exists. (AP0606)  <b>Fail:</b> An implementation for a handler registered for the POSIX-defined <i>SIGQUIT</i> signal does not exist. (AP0606)		<b>Source code declaration may look similar to:</b>  #include <signal.h>  void quitHandler( int signal )  <b>where:</b>  <i>signal.h</i>  Contains POSIX signal numbers such as <i>SIGQUIT</i> .  <i>int signal</i> Function called when the signal is raised.  <i>sigaction()</i> is the method provided by the AEP for an application to register for signal handling.	
End of Test				

Test Recording Log – APP_TC_042		
SAD File:		
Step 1 (source code file having <i>SIGQUIT</i> handler)	Step 2 (handler for <i>SIGQUIT</i> signal – Y/N?)	



## Test Summary APP\_TC\_042

Once testing is complete for every component of the application 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):**

AP0606 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.41APP\_TC\_043 – Registering with the Naming Service

### Test Case Number: APP\_TC\_043

Resource and ResourceFactory

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0709	Upon execution of a software module by the create operation, a Resource or a ResourceFactory component shall register with the Naming Service.

### References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 05-15-2006	Page 3-29, Section 3.1.3.2.2.5.1.3
SCA AppendixD	Version 2.2.2 05-15-2006	

### Test Objective

This test case verifies requirement AP0709. The objective of this test case is to verify that any module that is created during the execution of an application is registered successfully with the Naming Service.

### Places to Verify

Applications that implement a Resource or a ResourceFactory component register with the Naming Service.

### IDL References

None

### Preconditions

- The application source code files are available.

### Test Description

For every Resource (not started via a ResourceFactory) and ResourceFactory component in the application,

- A. Verify in the source code that a *bind* or *bind\_new\_context* operation exists to register the component name with the Naming Service.
1. **Pass:** A *bind* or *bind\_new\_context* operation exists to register the component name with the Naming service.
  2. **Fail:** A *bind* or *bind\_new\_context* operation does not exist to register the component name with the Naming service.
  3. **Fail:** Some other operation exists to register the component name with the Naming service.

## 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.

APP_TC_043				
Steps	Expected Results	Actual Results	Comments	Test Result
For every Resource (not started via a ResourceFactory) and ResourceFactory component in the application,				
A. Verify in the source code that a <i>bind</i> or <i>bind_new_context</i> operation exists to register the component name with the Naming Service. (AP0709)				
1. For registering purposes determine the ID used for the Naming Service.	<b>Pass:</b> The ID for the naming service is found. (AP0709)  <b>Fail:</b> The ID for the naming service is not found. (AP0709)			
2. Examine each source code and verify that a <i>bind</i> or <i>bind_new_context</i> operation exists to register the component name with the Naming Service	<b>Pass:</b> Each component should be registered using the <i>bind</i> or <i>bind_new_context</i> operation. (AP0709)  <b>Fail:</b> No other operator may be used to register a component's name with the Naming Service. (AP0709)		Source code statement looks similar to: <b>namingContext_var-&gt;bind (bindingName_var.in(), xxxxx_var.in() ACE_ENV_ARG_PARAMETER)</b> <b>;</b> <b>ACE_TRY_CHECK;</b>	
End of Test				

Test Recording Log – APP_TC_043		
Preliminary Step (Source code for Resources and ResourceFactory components)	Step 1 (ID for naming service found?)	Step 2 (bind or bind_new_context Operation Exists?)

## Test Summary APP\_TC\_043

Once testing is complete for every component of the application 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):**

AP0709 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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

## B.42 APP\_TC\_044 – Usage of stringified IORs

**Test Case Number:** APP\_TC\_044

Applications::CORBA Services

### Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
AP0740	Applications shall not utilize static stringified IORs.

### References

Document Name	Version/Date	Location
SCA	Version 2.2.2 05-15-2006	Page 3-95, Section 3.2.1.2

### Test Objective

This test case verifies AP0740. The objective of this test is to verify that the application does not use static stringified IOR's. Static stringifying of an IOR is accomplished by assigning (hard-coding) a static string value within the application – the value will always remain the same.

### Places to Verify

Applications that implement stringified IOR's.

### IDL References

None

### Preconditions

The application source code files are available.

### Test Description

- A. Verify that the application is not using *statically stringified IOR* (AP0740).
  1. **Pass:** Application is not using statically stringified IOR.
  2. **Fail:** Application is using statically stringified IOR.

## 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.

APP_TC_044				
Steps	Expected Results	Actual Results	Comments	Test Result
<b>A. Verify that the application is not using <i>statically stringified</i> IOR (AP0740).</b>				
1. Examine the source code for the naming context IOR and verify that the IOR is not statically stringified. (AP0740)	<b>Pass:</b> Application is not using statically stringified IOR. (AP0740)  <b>Fail:</b> Application is using statically stringified IOR. (AP0740)		May need the help of a software developer to locate the source code files.  A statically stringified IOR would appear as a string literal in the program.  A dynamically created stringified IOR would be retrieved from the NameService, a file, or a command line argument.	
<b>End of Test</b>				



Test Recording Log: APP_TC_044				
Step 1 (source files with Naming Context IOR)				

## Test Summary APP\_TC\_044

Once testing is complete for every component of the application 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):**

AP0740 \_\_\_\_\_

**Failed Items (Section/StepNumber):**

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

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

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

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