Joint Tactical Networking Center Test and Evaluation Laboratory

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

Appendix B Volume 5
Test Procedures for AEP and Miscellaneous Requirements

13 April 2022

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

Prepared by:
Joint Tactical Networking Center Test and Evaluation Laboratory

TABLE OF CONTENTS

APPENDIX B VOLUME 5 TEST PROCEDURES

B.5 Volume 5 AEP and Miscellaneous Requirements

This volume consists of manual test cases which verify requirements related to Software Communication Architecture (SCA) Application Environment Profile (AEP) and miscellaneous requirements not covered by the criteria of the other Appendix B volumes.

B.5.1. OE_TC_001 - CORBA :: NamingService

Test Case Number: OE_TC_001
CORBA::NamingService

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0007	The "kind" element of each NameComponent shall be "" (null string).
OE0746	The OE shall provide an implementation of a CORBA *Naming service* which implements the *CosNaming* module NamingContext interface operations: *bind*, *bind_new_context*, *unbind*, *destroy*, and *resolve* as defined in the OMG Interoperable *Naming service* Specification [6] using the IDL found in Appendix A of that reference.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	P 3-2
OMG interoperable *Naming service* Specification	Version 1.0.1 November 2000	P. 2-7, Section 2.2.3.1; P 2-8, Section 2.2.4, P 2-9, Sections 2.2.5, 2.2.6.2, and 2.2.7
OMG interoperable *Naming service* Specification, Appendix A.	Version 1.0.1 November 2000	Pg. A-19 thru A-20

Test Objective

This test case verifies OE0007 and OE0746. The objective of this test is to verify that the OE provides an implementation of a CORBA *Naming service* and implements the *CosNaming* module *NamingContext* interface operations: *bind*, *bind_new_context*, *unbind*, *destroy* and *resolve*.

Places to Verify

Core Framework.

IDL References

```
module CosNaming  {
        typedef string Istring;

                struct NameComponent {
                        Istring id;
                        Istring kind;
                };
        typedef sequence<Namecomponent> Name;

        void bind(in Name n, in Object obj) raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
        NamingContext bind_new_context(in Name n) raises(NotFound, AlreadyBound, CannotProceed, InvalidName);
        void unbind(in  Name n) raises(NotFound, CannotProceed, InvalidName);
        void destroy() raises(NotEmpty);
        Object resolve (in Name n) raises(NotFound, CannotProceed, InvalidName);
                }
```

Preconditions

- The source code and any IDL files are available. At a minimum, the header files (c/c++ .h files or the equivalent) are necessary.
- The documentation for the naming service is available.

Test Description

Note: This test only verifies the signature of the naming service. Since commercial source code is normally proprietary and not made available for verification, functionality is not tested. The documentation is reviewed to see if the developer says that the functionality is provided.

A. Identify the *Naming service* implementation (OE0007, OE0746).
   1. **Fail**: The *Naming service* implementation cannot be identified.

B. Verify that the *Naming service* implementation of the *CosNaming* module exists and conforms to the OMG IDL specification (OE0746).
   1. **Pass:** The *CosNaming* module exists and conforms to the OMG IDL specification.
   2. **Fail**: The *CosNaming* module does not exist.
   3. **Fail**: The *CosNaming* module does not conform to the OMG IDL specification.

C. Verify that the implemented *CosNaming* module includes the interface operations, bind, *bind_new_context*, *unbind*, *destroy* and *resolve*, and that they conform to the IDL from OMG interoperable *Naming service* Specification, Appendix A (OE0746).
   1. **Pass:** The *CosNaming module* includes all the above the interface operations and their signatures are implemented as defined in the OMG.
   2. **Fail**: The *CosNaming module* does not include one or more of the interface operations: *bind, bind_new_context, unbind, destroy* and *resolve* as defined in the OMG.
   3. **Fail**: The interface operations: *bind, bind_new_context, unbind, destroy* and *resolve* do not follow the signatures defined in the OMG.

D. Verify that the kind element of each NameComponent structure is a null string. (OE0007)
   1. **Pass**: the kind element is a string and a null value is accepted.
   2. **Fail**: The kind element is not a string.
   3. **Fail**: A null string is not accepted.

Manual Test Steps

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

OE_TC_001				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the *Naming service* implementation. (OE0007, OE0746)				
1. Locate and record the source code file directory(s) for the implementation of the Naming Service.	**Pass**: The directory(s) are located. (OE0007, OE0746) **Fail**: the directory(s) are not located. (OE0007, OE0746)		There may be more than one directory. Usually these will be just the header files (c/c+= .h files or equivalent). An IDL, if provided should be verified also. The help of the software engineer may be necessary to locate the directories and source code files.	
B. Verify that the *Naming service* implementation of the *CosNaming* module exists and conforms to the OMG IDL specification (OE0746).				

2. Examine the source code files and verify that the *CosNaming* module is implemented and conforms to the IDL specification.	**Pass**: The *CosNaming module* is implemented and conforms to the IDL specification. (OE0746)
	
	Fail: The *CosNaming module* does not exist. (OE0746)
	
	Fail: The *CosNaming module* does not conform the OMG IDL Specification. (OE0746)
			IDL implementation is:
			
			module *CosNaming* {
			…
			}	

C. Verify that the implemented *CosNaming* module includes the interface operations, *bind, bind_new_context, unbind, destroy* and *resolve,* and that they conform to the OMG IDL specification. (OE0746)				

OE_TC_001				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Examine the source code files and verify that the *CosNaming module* interface *bind* operation is implemented and conforms to the IDL specification.	**Pass**: The *bind* operation is implemented in accordance with the OMG IDL specification. (OE0746) **Fail**: the *bind* operation does not exist. (OE0746) **Fail**: The *bind* operation does not follow the signature in the OMG IDL Specification. (OE0746)		IDL implementation is: **void *bind***(in Name n, in Object obj) raises(NotFound, CannotProceed, InvalidName, AlreadyBound);	
4. Examine the source code files and verify that the *CosNaming module* interface *bind_new_context* operation is implemented and conforms to the IDL specification.	**Pass**: The *bind_new_context* operation is implemented in accordance with the OMG IDL specification. (OE0746) **Fail**: the *bind_new_context* operation does not exist. (OE0746) **Fail**: The *bind_new_context* operation does not follow the signature in the OMG IDL Specification. (OE0746)		IDL implementation is: **NamingContext *bind_new_context***(in Name n) raises(NotFound, AlreadyBound, CannotProceed, InvalidName);	
5. Examine the source code files and verify that the *CosNaming module* interface *unbind* operation is implemented and conforms to the IDL specification.	**Pass**: The *unbind* operation is implemented in accordance with the OMG IDL specification. (OE0746) **Fail**: the *unbind* operation does not exist. (OE0746) **Fail**: The *unbind* operation does not follow the signature in the OMG IDL Specification. (OE0746)		IDL implementation is: **void *unbind***(in Name n) raises(NotFound, CannotProceed, InvalidName);	

OE_TC_001				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Examine the source code files and verify that the *CosNaming module* interface *destroy* operation is implemented and conforms to the IDL specification.	**Pass**: The *destroy* operation is implemented in accordance with the OMG IDL specification. (OE0746) **Fail**: the *destroy* operation does not exist. (OE0746) **Fail**: The *destroy* operation does not follow the signature in the OMG IDL Specification. (OE0746)		IDL implementation is: **void *destroy*** () raises(NotEmpty);	
7. Examine the source code files and verify that the *CosNaming module* interface *resolve* operation is implemented and conforms to the IDL specification.	**Pass**: The *resolve* interface operation is implemented in accordance with the OMG IDL specification. (OE0746) **Fail**: the *resolve* operation does not exist. (OE0746) **Fail**: The *resolve* operation does not follow the signature in the OMG IDL Specification. (OE0746)		IDL implementation is: **Object *resolve*** (in Name n) raises(NotFound, CannotProceed, InvalidName);	
D. Verify that the kind element of each NameComponent structure is a null string. (OE0007)				
8. Find every invocation of bind() and bind_new_context().	**Pass**: All invocations of the named functions are identified (OE0007)		void bind (in CosNaming::Name n, in Object obj) CosNaming::NamingContext bind_new_context (in CosNaming::Name n)	

OE_TC_001				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
9. Verify that the input Name is set up properly.	**Pass**: The Name parameter is properly set up. (OE0007) **Fail**: The Name parameter is not properly set up. (OE0007)			
End of Test				

Test Recording Log – OE_TC_001		
Step	**Test**	**Comments**
1	Location (source code) of Naming Service.	
2	Cos Naming Module exists and follows OMG spec.	
3	*bind* operation follows OMG	
4	*bind_new_context* follows OMG	
5	*unbind* follows OMG	
6	*Destroy* follows OMG	
7	*resolve* follows OMG	
8	Kind element is a string value and a null string is allowed	
9	section 2.2.3.1 of the OMG specification (bind) is implemented.	

Test Recording Log – OE_TC_001		
Step	**Test**	**Comments**
10	section 2.2.4 (resolve) is implemented.	
11	section 2.2.5 of the OMG specification (unbind) is implemented.	
12	section 2.2.5 of the OMG specification (unbind) is implemented.	
13	section 2.2.7 of the OMG specification (destroy) is implemented.	

Test Summary - OE_TC_001

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

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

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

OE0007_____

OE0746_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.2. OE_TC_002 - CORBA :: Log Producer

Test Case Number: OE_TC_002

CORBA::Log Producer

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0011	A log producer shall only output log records that contain an enabled CosLwLog::LogLevel value
OE0012	Log producers shall use their component identifier attribute in the producerId field of the CosLwLog::ProducerLogRecord.
OE0013	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.
OE0747	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 CosLwLog::LogLevelSequence.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2; 05-15-2006	Page 3-2, Sections 3.1.2.2.1
Lightweight Log Service Specification	Version 1.1, February 2005	Page 3-2, Section 3.2.2; Page 3-5, Section 3.2.8 Page 3-20 to 3-22, Section 3.3.3

Test Objective

This test case verifies OE0011, OE0012, OE0013 and OE0747. The objective of this test is to verify that the OE outputs proper log records.

Places to Verify

Application. DomainManager and DeviceManager

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

- The source code files are available.
- The Lightweight Log service is implemented in the OE.

Test Description

A. Locate all the source code in the OE that implements the *write_record* or *write_records* operations. (OE0011, OE0012, OE0013, OE0747)
   1. **Untested**: Unable to locate the *write_record* or *write_records* operations.

B. For each *write_record* or *write_records* operation identified in Step A, verify the following:
   1. Verify the component identifier attribute is stored in the producerId field of the Log Record. (OE0012)
      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. (OE0013)
   a. **Pass:** The component verifies the existence of a log service before attempting to use it.
   b. **Pass:** The component handles any exception resulting from the lack of a log service and continues to operate normally.
   c. **Fail:** The component does not operate properly in the absence of a log service.
3. Verify that only records with enabled LogLevel values are output. (OE00011)
   a. **Pass:** Only Log messages with an 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" and a value that is CosLwLog::LogLevelSequence, is implemented. (OE0747)
1. **Pass:** The proper *configure* property is implemented.
2. **Fail:** The *configure* property is not implemented.

Semi-automated Test Steps

After running JTAP's Application PRODUCER_LOG_LEVEL, DeviceManager PRODUCER_LOG_LEVEL or DomainManager PRODUCER_LOG_LEVEL tests, perform manual steps 1 through 5. OE0013 is always tested manually.

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 recording of data.

OE_TC_002				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all the source code in the OE that implements the *write_record* or *write_records* operations. (OE0011, OE0012, OE0013, OE0747)				
NOTE: The OE software engineer can be of assistance in locating the source code for the log producing components. A search engine such as grep or a development tool such as Visual C++ can be used to search all the source code to find specific code to be examined. Do not depend on Microsoft explorer's search operation, as it can be demonstrated that it will not find a string within a file.				
1. Locate and record files that invoke the *write_record* or *write_records* operations.	**Untested**: No *write_record* or *write_records* operations are found. (OE0011, OE0012, OE0013, OE0747)			
B. For each *write_record* or *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. (OE0012)				

2. Locate the producerId field in the log record and verify that it is set to the component identifier.	**Pass**: producerId field in the log record is set to the component identifier. (OE0012)
	
	Fail: producerId field in the log record is not set to the component identifier. (OE0012)
			This could be done as a preset or dynamically, depending on how log records are produced.
			struct ProducerLogRecord {
			string **producerId**;
			string producerName;
			LogLevel level;
			string logData;
			}	

OE_TC_002				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
2. Verify that the component works properly when it is not connected to a log service. (OE0013)				
3. Verify that the component does not fail if there is no log service.	**Pass**: The component does not fail in the absence of a log service. (OE0013) **Fail**: The component fails in the absence of a log service. (OE0013)		There are several ways to handle this situation. I.E. Catching an exception resulting from no log service or verifying that the log service object exists before trying to use it are two ways.	
3. Verify that only records with enabled LogLevel values are output. (OE00011)				
4. Verify that the log record(s) passed to *write_record* and *write_records* are of type CosLwLog::ProducerLogRecord.	**Pass**: The log record(s) are of type CosLwLog::ProducerLogRecord. (OE0011) **Fail**: The log record(s) are not of type CosLwLog::ProducerLogRecord. (OE0011)		CosLwLog::ProducerLogRecord is a struct that looks like this **struct ProducerLogRecord** **{** **string producerId;** **string producerName;** **LogLevel level;** **string logData;** **};**	
5. Verify that the LogLevel is checked and that *write_record* or *write_records* are called only for enabled log levels.	**Pass**: The log service's *write_record* and/or *write_records* operations are called only for enabled log levels. (OE0011) **Fail**: The log service's *write_record* and/or *write_records* operations are not called for enabled log levels. (OE0011)		The checking of LogLevel should check the same data that is being saved by the configure operation (see step 8)	
C. Verify that a configure property, with an id of "PRODUCER_LOG_LEVEL" and a value that is CosLwLog::LogLevelSequence, is implemented. (OE0747)				

OE_TC_002				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Locate the *configure* operation of the component and verify it implements a property with an id of "PRODUCER_LOG_LEVEL" and only valid values are allowed.	**Pass**: The property exists and input values are validated before they are set by the *configure* operation. (OE0747) **Fail**: Either the property does not exist or input values are not validated before they are set by the *configure* operation. (OE0747)		Log Level values discussed in the OMG LWL Specification include the following: SECURITY_ALARM = 1; FAILURE_ALARM = 2; DEGRADED_ALARM =3; EXCEPTION_ERROR =4; FLOW_CONTROL_ERROR =5; RANGE_ERROR =6; USAGE_ERROR = 7; ADMINISTRATIVE_EVENT = 8; STATISTIC_REPORT = 9; // Values ranging from 10 to 26 are reserved for // 17 debugging levels.	
End of Test				

Test Recording Log – OE_TC_002						
Step 1 (Files writing records)	**Step 2** (Log producing Component)	**Step 3** (No log service)	**Step 4** (log record(s) type)	**Step 5** (Log Level)	**Step 6** (configure operation)	**Notes**

Test Summary OE_TC_002

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

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

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

OE0011_____

OE0012_____

OE0013_____

OE0747_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.3. OE_TC_003 - Event Service

Test Case Number: OE_TC_003

Event Service

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0066	The Incoming Domain Management Channel name shall be "IDM_Channel".
OE0067	The Outgoing Domain Management Channel name shall be "ODM_Channel".
OE0702	The OE shall provide two standard event channels: Incoming Domain Management and Outgoing Domain Management.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3.2 thru 3-3, Section 3.1.2.3.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	C-21

Test Objective

This test case verifies OE0066, OE0067, and OE0702. The objective of this test is to verify that two event channels, named IDM_Channel and ODM_Channel, are created.

Places to Verify

Devices, DomainManager

IDL References

Operations

```
void registerWithEventChannel ( in Object registeringObject, in string registeringId, in string eventChannelName )
        raises (CF::InvalidObjectReference,
                CF::DomainManager::InvalidEventChannelName,
                CF::DomainManager::AlreadyConnected);
```

Preconditions

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

Test Description

A. Verify that the Domain ManagerConfiguration Descriptor (DMD) file exists. (OE0702)
   1. **Pass:** A DMD file exists.
   2. **Fail:** A DMD file does not exist.

B. Verify that the Event Channels *IDM_Channel* and *ODM_Channel* are referenced in the DMD file. (OE0702)
   1. **Pass:** Both event channels are referenced in the DMD file.
   2. **Fail:** Both event channels are not referenced in the DMD file.

C. Verify that the Software Package Descriptor (SPD) file is referenced in the DMD file. (OE0702)
   1. **Pass:** A SPD file is referenced in the DMD.
   2. **Fail:** A SPD file is not referenced in the DMD.

D. Verify that the Software Component Descriptor (SCD) file is referenced in the SPD file. (OE0702)
   1. **Pass:** A SCD file is referenced in the SPD.
   2. **Fail:** A SCD file is not referenced in the SPD.

E. If a Software Component Descriptor (SCD) file is referenced in the SPD, verify that 2 ports with names of *IDM_Channel* and *ODM_Channel* are referenced. (OE0066, OE0067)
   1. **Pass:** Ports exist with names of "IDM_Channel" and "ODM_Channel".
   2. **Fail:** Ports do not exist with names of "IDM_Channel" and "ODM_Channel".

F. Verify that a proper event channel is registered using the *registerWithEventChannel* operation. (OE0066, OE0067)
   1. **Pass:** *ODM_Channel* and *IDM_Channel* event channels are registered.
   2. **Fail:** *ODM_Channel* and *IDM_Channel* event channels are not registered.

Manual Test Steps

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

OE_TC_003				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the Domain Manager Configuration Descriptor (DMD) file exists. (OE0702)				
1. Locate the DMD file	**Pass:** A DMD file exists. (OE0702) **Fail:** A DMD file does not exist. (OE0702)		DMD declaration look similar to **DomainManager.dmd.xml**	
B. Verify that the Event Channels *IDM_Channel* and *ODM_Channel* are referenced in the DMD file. (OE0702)				
2. In the DMD file, locate the IDM Channel name and record the name of the file.	**Pass:** An IDM Channel name is found. (OE0702) **Fail:** An IDM Channel name is not found. (OE0702)		<service> <usesidentifier>IDM_Channel</usesidentifier> <findby> <domainfinder type="eventchannel" name="**IDM_Channel**"/> </findby> </service>	
3. In the DMD file, locate the ODM Channel name and record the name of the file.	**Pass:** An ODM Channel name is found. (OE0702) **Fail:** An ODM Channel name is not found. (OE0702)		<service> <usesidentifier>ODM_Channel</usesidentifier> <findby> <domainfinder type="eventchannel" name="**ODM_Channel**"/> </findby> </service>	
C. Verify that the Software Package Descriptor (SPD) file is referenced in the DMD file. (OE0702)				

OE_TC_003				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Locate the SPD file in the DMD found in Step 1.	**Pass:** A SPD file is referenced in the DMD. (OE0702) **Fail:** A SPD file is not referenced in the DMD. (OE0702) Write down the name of the SPD file in the Test Recording Log		SPD declaration look similar to <domainmanagersoftpkg> <localfile name="**DomainManager.spd.xml**"/> </domainmanagersoftpkg>	
D. Verify that the Software Component Descriptor (SCD) file is referenced in the SPD file. (OE0702)				
5. Locate the SCD file found in the SPD.	**Pass:** A SCD file is referenced in the SPD. (OE0702) **Fail:** A SCD file is not referenced in the SPD. (OE0702) This step is another method to verify that the event channels exist. Write down the name of the SCD file in the Test Recording Log.		SCD declaration look similar to <descriptor> <localfile name="**DomainManager.scd.xml**"/> </descriptor>	
E. If a Software Component Descriptor (SCD) file is referenced in the SPD, verify that 2 ports with names of *IDM_Channel* and *ODM_Channel* are referenced. (OE0066, OE0067)				

OE_TC_003				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Locate the *usesname* element in the SCD file that defines the *IDM_Channel* and *ODM Channel* event channels.	**Pass:** Ports exist with names of "IDM_Channel" and "ODM_Channel". OE0066, OE0067) **Fail:** Ports do not exist with names of "IDM_Channel" and "ODM_Channel". OE0066, OE0067) *usesname* elements references the name of the *IDM_Channel* and *ODM_Channel*. Write the values of the usesnames element in the Test Recording Log.		Port declaration look similar to <componentfeatures> <supportsinterface repid="IDL:CF/DomainManager:2.2" supportsname="CF::DomainManager"/> <ports> <uses repid="xxxxx" usesname="**IDM_Channel**"/> <uses repid="yyyyy" usesname="**ODM_Channel**"/> </ports> </componentfeatures>	
F. Locate *registerWithEventChannel* operation in the source code. (OE0066, OE0067)				
7. In the SPD file, from Step 4, and determine the source code for the log service.	Source code is determined and located.		The name of the executable in the SPD file is the best clue for the name of the source code.	
8. Identify the source code for *registerWithEventChannel* operation.	**Pass:** *registerWithEventChannel* operation found in the source code. OE0066, OE0067) **Fail:** *registerWithEventChannel* operation is not found in the source code. OE0066, OE0067)			

OE_TC_003				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
9. Verify that the proper event channel names are passed into the *registerWithEventChannel* operation.	**Pass:** *ODM_Channel* and *IDM_Channel* event channels are registered. (OE0066, OE0067) **Fail:** *ODM_Channel* and *IDM_Channel* event channels are not registered. (OE0066, OE0067)		void registerWithEventChannel (in Object registeringObject, in string registeringId, in string **eventChannelName**)	
End Of Test				

Test Recording Log – OE_TC_003				
Step 1 (DMD File)				
Step 2 (IDM Channel name)				
Step 3 (ODM Channel name)				
Step 4 (SPD file)				
Step 5 (SCD file)				
Step 6 (usesname)				

Test Summary – OE_TC_003

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

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

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

OE0066 _____

OE0067 _____

OE0702 _____

Failed Items (Section/Step Number):

_____

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.5.4. OE_TC_021 - AEP Applications have no abnormal termination

Test Case Number: OE_TC_021

Application abnormal termination

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0661	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)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 1-4, Section 1.4.
SCA Appendix B: Application Environment Profile (AEP)	Version 2.2.2 FINAL / 15 May 2006	Pages 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.	10 September 2004	Pages 43, Section 6.3.1

Test Objective

This test case verifies OE0661. Table B-6 of the AEP Amended specifies the mandatory POSIX Signals function to be provided by the OE. Additionally, the SCA AEP Amended specifies that the signals generated by these signal functions when called must be handled properly so that other processes are not impacted because certain RTOSs do not support multiple processes. The objective of this test case is to verify that the OE, which is the provider of these signals functions, implements signals handlers for each signal generated by the signal functions in order to properly manages process termination.

For product lines that implement the AEP Amended, OE0798 is evaluated in lieu of OE0661 for the same objective in the test case OE_TC_170 found in Appendix C of this MOESTD.

Places to Verify

OE core framework

IDL References

None

Preconditions

- Requirement OE0660 (test case, OE_TC_131) executed and passed.
- The source code of the Core Framework is available.

Test Description

A. Identify the source code and/or OE Header files that implement POSIX Signal functions. (OE0661)
   1. **Pass:** The source code and/or OE Header files are available.
   2. **Untested**: The source code and/ or OE Header files are not available. If untested, then stop test.

For each POSIX Signal functions in Table B-6 of the AEP, perform the following:

B. Verify there is a signal handler for each of the POSIX signal functions. (OE0661)
   1. **Pass:** All signal functions are registered with its corresponding signal handler.
   2. **Fail:** One or more signal function(s) do not register with a 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 recording of data.

AP_TC_021				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code and/or OE Header files that implement POSIX Signal functions. (OE0661)				
1. Perform a search for header files for each signal function. Refer to list of functions in the Comments cell.	**Pass:** The source code and/or OE Header files are available for all functions. (OE0661) **Untested**: The source code files are not available for one or more function(s). (OE0661)		abort() kill() pause() raise() sigaction() sigaddset() sigdelset() sigemptyset() sigfillset() sigismember() signal() sigpending() sigprocmask() sigsuspend() sigwait()	
For each POSIX Signal functions in Table B-6 of the AEP, perform the following:				
B. Verify there is a signal handler for the POSIX signal functions. (OE0661)				

AP_TC_021				
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.	**Pass:** Signal handlers are implemented for each POSIX signal. (OE0661) **Fail:** One or more signal(s) do not have a signal handler. (OE0661)		The OE 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. Therefore, it is expected for the OE to provide these handlers for processes to register since it is the OE that provides the signal functions.	
End Of Test				

Test Recording Log OE_TC_021	
Step 1 (source file having POSIX signal functions)	**Step 2** (List signal hander for each signal function.

Test Summary OE_TC_021

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

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

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

OE0661_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.5. OE_TC_044 - AEP file mode creation masks

Test Case Number: OE_TC_044

AEP File mode creation mask

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0666	An application that conforms to the AEP shall be guaranteed that the file mode creation mask for any object created by any process is S-IRWXU; that is, the object shall be fully accessible to the creator.

References

Document Name	Version/Date	Location (Pages, Section)
SCA Appendix B: AEP	Version 2.2.2 15 May 2006	Page B-7, Sec. B.4.1.8

Test Objective

This test case verifies OE0666. The objective of this test is to verify that the file mode creation mask for any object created is at a minimum S-IRWXU. This mask says that the creator, i.e., owner, will have read, write, and execute rights to the object created. The file mode creation mask is used for the creation of directories and files.

Places to Verify

FileSystem's and FileManager

IDL References

Operations

void mkdir (in string directoryName) raises (CF::InvalidFileName, CF::FileException);
CF::File create (in string fileName) raises (CF::InvalidFileName, CF::FileException);

Preconditions

- The FileSystem and FileManager source code files are available.

Test Description

A. Locate the source code for the create operation of the FileSystem(s) and the FileManager. (OE0666)

1. **Untested**: The create operation cannot be found.

B. Verify that the file mode creation mask for the create operation is at a minimum S-IRWXU. (OE0666)
   1. **Pass:** The file mode creation mask is at a minimum S-IRWXU.
   2. **Fail:** The file mode mask does not include S-IRWXU.

C. Locate the source code for the mkdir operation of the FileSystem(s) and the FileManager. (OE0666)
   1. **Untested**: The mkdir operation cannot be found.

D. Verify that the file mode creation mask for the mkdir operation is at a minimum S-IRWXU. (OE0666)
   1. **Pass:** The file mode creation mask is at a minimum S-IRWXU.
   2. **Fail:** The file mode mask does not include S-IRWXU.

Manual Test Steps

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

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

OE_TC_044				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the create operation of the FileSystem(s) and the FileManager. (OE0666)				
1. Locate the source code for the create operation in the FileSystem(s) and FileManager and record the file names.	**Untested**: The source code cannot be located.(OE0666)			
B. Verify that the file mode creation mask for the create operation is at a minimum S-IRWXU. (OE0666)				
2. Verify that the file mode creation mask for the create operation will include S-IRWXU.	**Pass**: The file mode creation mask includes S-IRWXU. (OE0666) **Fail**: The file mode creation mask does not include S-IRWXU. (OE0666)		The File mode creation mask is frequently a mask made of logically combined values. Ensure that the and'ing and or'ing does results in the owner of the object having read, write, and execute rights to the object being created. Typically the mask consists of 3 octal digits, with the first digit being the owner permission, the second being group permission and the third being world (everyone else) permission. So the octal mask expected would be 7xx where the xx bits do not matter.	
C. Locate the source code for the mkdir operation of the FileSystem(s) and the FileManager. (OE0666)				
3. Locate the source code for the mkdir operation in the FileSystem(s) and FileManager and record the file names.	**Untested**: The source code cannot be located.(OE0666)			

OE_TC_044				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
D. Verify that the file mode creation mask for the mkdir operation is at a minimum S-IRWXU. (OE0666)				
4. Verify that the file mode creation mask for the mkdir operation will include S-IRWXU.	**Pass**: The file mode creation mask does not include S-IRWXU. (OE0666) **Fail**: The file mode creation mask does not include S-IRWXU. (OE0666)		The File mode creation mask is frequently a mask made of logically combined values. Insure that the and'ing and or'ing does results in the owner of the object having read, write, and execute rights to the object being created. Typically the mask consists of 3 octal digits, with the first digit being the owner permission, the second being group permission and the third being world (everyone else) permission. So the octal mask expected would be 7xx where the xx bits do not matter.	
End Of Test				

Test Recording Log – OE_TC_044			
Step 1 (create file names)	Step 2 (valid mask)	Step 3 (mkdir file names)	Step 4 (valid mask)

Test Summary OE_TC_044

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

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

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

OE0666_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.6. OE_TC_059 - Mandatory interfaces of the AEP

Test Case Number: OE_TC_059

AEP mandatory functions and options

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0001	The OE shall provide the functions and options designated as mandatory by the AEP defined in Appendix B.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-1, Section 3.1.1
SCA Appendix B: AEP	Version 2.2.2 15 May 2006	Page B-2 thru B18, Section B.4

Test Objective

This test case verifies OE0001. The objective of this test is to verify that the OE provides, at a minimum, the options and functions designated as mandatory in Appendix B of the SCA.

Places to verify

OE and OS documentation.

IDL References

None.

Preconditions

- The OE and OS documentation is available.

Test Description

A. Verify in the OE and OS documentation that the Appendix B mandatory options and functions are provided. (OE0001)
   1. **Fail**: One or more Appendix B mandatory options or functions are missing.
   2. **Pass**: All Appendix B mandatory options and functions are provided.

Manual Test Steps

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

OE_TC_059				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify in the OE documentation that the Appendix B mandatory functions are provided. (OE0001)				
1. Identify the design and version documents supplied by the developer.	Design documents are identified.		This would include such documents as the System Design Document (SDD), Version Description Document (VDD), System Requirements Specification (SRS) and any other documentation that may have details on the OS being used.	
2. In the documentation identify the underlying OS, its version, and the documentation for it supplied by the developer.	OS, version, and documentation are identified.		The AEP functions are normally supplied by the OS.	
3. Verify in the OS documentation that all the mandatory options and functions listed in Appendix B of the SCA are supplied.	**Pass**: All the AEP mandatory functions are supplied. (OE0001) **Fail**: One or more mandatory options or functions are not supplied. (OE0001)		This may include visiting web sites, such as: http://get.posixcertified.ieee.org/cert_prodlist.tpl, to verify certification rather than inspecting for each option and function.	
End Of Test				

Test Recording Log – OE_TC_059		
Step 1 (OE design documents)	**Step 2** (OS name, version, and documentation)	**Step 3** (Documentation includes all AEP mandatory operations)

Test Summary OE_TC_059

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

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

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

OE0001_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.7. OE_TC_061 - Minimum CORBA

Test Case Number: OE_TC_061

OE::CORBA

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0003	The OE shall include middleware that, at a minimum, provides the services and capabilities of minimum CORBA as specified by the OMG Document in reference [5].

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-1, Section 3.1.2 Page 3-97, Section 3.3.2
Minimum CORBA Specification	Specification Version 1.0: OMG Document formal/02-08-01, August 2002	Page 1-11 thru 1-28

Test Objective

This test case verifies OE0003. The objective of this test is to verify that the functionality of minimum CORBA, as described in Minimum CORBA Specification Version 1.0: OMG Document formal/02-08-01, August 2002, is met by the OE middleware.

Places to Verify

OE and ORB documentation.

IDL References

None.

Preconditions

- The OE and ORB documentation is available.

Test Description

A. Determine in the OE documentation the ORB and the version number of the ORB that is used to provide the minimum CORBA requirements. (OE0003)

1. **Pass**: the ORB and its version number are determined.
2. **Untested**: The ORB cannot be determined.
3. **Untested**: the version of the ORB cannot be determined.

B. Verify in the ORB documentation that the ORB implemented supports the minimum CORBA requirements. (OE0003)
   1. **Pass**: ORB does support minimum CORBA.
   2. **Fail**: ORB does not support minimum CORBA.

Manual Test Steps

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

OE_TC_061				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Determine in the OE documentation the ORB and the version number of the ORB that is used to provide the minimum CORBA requirements. (OE0003)				
1. Determine from the OE documentation the ORB implementation.	The ORB name is determined.		The SRS, SDD and SVD or equivalent are the best sources for this information.	
2. Verify the ORB version being used.	**Pass**: The ORB provides minimum CORBA functionality. (OE0003) **Fail**: The ORB does not provide minimum CORBA functionality. (OE0003)			
B. Verify in the ORB documentation that the ORB implemented supports the minimum CORBA requirements. (OE0003)				
3. Verify in the ORB documentation that the minimum CORBA operations are provided.	**Pass**: All required minimum CORBA operations are present. (OE0003) **Fail**: One or more required minimum CORBA operations are missing. (OE0003)		This documentation could include a letter of certification from the ORB vendor. The ORB documentation will probably be available only online.	
End Of Test				

Test Recording Log – OE_TC_061	
Step 1 (ORB)	
Step 2 (ORB version)	
Step 3 (ORB supports minimum CORBA)	

Test Summary OE_TC_061

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

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

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

OE0003_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.5.8. OE_TC_063 - CORBA :: CosEventComm

Test Case Number: OE_TC_063

CORBA::CosEventComm

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0063	A component (e.g., Resource, DomainManager, etc.) that consumes events shall implement the CosEventComm PushConsumer interface.
OE0064	A component (e.g., Resource, Device, DomainManager, etc.) that produces events shall implement the CosEventComm PushSupplier interface and use the CosEventComm PushConsumer interface for generating the events.
OE0065	A producer component shall not forward or raise any exceptions when the connection to a CosEventComm PushConsumer is a nil or invalid reference.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-2, Section 3.1.2.3.1
OMG Event Service Specification	Version 1.1 March 2001	Page 2-1 thru 2-2, Section 2.1

Test Objective

This test case verifies OE0063, OE0064, and OE0065. 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

Within the source code for producers and consumers.

IDL References

Exceptions

```
exception Disconnected{};
```

Operations

```
interface PushConsumer
{
  void push (in any data) raises(Disconnected);
```

```
    void disconnect_push_consumer();
    };
interface PushSupplier
{
    void disconnect_push_supplier();
};
```

Preconditions

- The source code for the OE is available.

Test Description

A. Find each component that declares itself to be a PushConsumer and verify that the CosEventComm *EventConsumer* interface is implemented. (OE0063)
   1. **N/A**: No component declares itself to be a PushConsumer.
   2. **Fail**: A PushConsumer component does not implement the CosEventComm *PushConsumer* interface.
   3. **Pass**: The PushConsumer components implement the CosEventComm *PushConsumer* interface.

B. Find each component that declares itself to be a PushSupplier and verify that the CosEventComm *PushSupplier* is implemented. (OE0064)
   1. **N/A**: No component declares itself to be a PushSupplier.
   2. **Fail**: A PushSupplier components do not implement the CosEventComm *PushSupplier* interface.
   3. **Pass**: The PushSupplier components implement the CosEventComm *PushSupplier* interface.

C. Verify that the components identified in step A captures or handles exceptions when a NIL or invalid reference is used for the connection to the *PushConsumer*. (OE0065)
   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 NIL or invalid references and does not propagate them outside the component.

Semi-automated Test Steps

After running JTAP's DomainManager registerWithEventChannel unregisterFromEventChannel test, perform manual steps 1 through 6.

Manual Test Steps

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

OE_TC_063				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Find each component that declares itself to be a *PushConsumer* and verify that the CosEventComm *EventConsumer* interface is implemented. (OE0063)				
1. Search the source code for *PushConsumer*.	**N/A**: No component declares itself to be a *PushConsumer*. (OE0063) Skip step 2.			
2. Verify that the component implements the *push* and *disconnect_push_consumer* methods.	**Fail**: The *push* or the *disconnect*_push_consumer method is not implemented. (OE0063) **Pass**: The push and the *disconnect*_push_consumer method are implemented. (OE0063)			
B. Find each component that declares itself to be a *PushSupplier* and verify that the CosEventComm *PushSupplier* is implemented. (OE0064)				
3. Search the source code for *PushSupplier*.	**N/A**: No component declares itself to be a *PushSupplier*. (OE0064) Skip step 4.			
4. Verify that the component implements the *disconnect_push_supplier*(..) methods.	**Fail**: The *disconnect_push_supplier* method is not implemented. (OE0064) **Pass**: The *disconnect_push_supplier* method is implemented. (OE0064)			

OE_TC_063				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
C. Verify that the components identified in step B captures or handles exceptions when a NIL or invalid reference is used for the connection to the *PushConsumer*. (OE0065)				
5. Locate all invocations of the *push* method in the source code.				
6. Verify that the push method does not forward or raise an exception when the *PushConsumer* object is a nil or invalid reference.	**Pass**: No exception is raised or forwarded when the *PushConsumer* object is a nil or an invalid reference. (OE0065) **Fail**: An exception is not handled by the *push* method. (OE0065) **Fail**: The *push* method raises an exception when the *pushConsumer* object is nil or invalid. (OE0065)			
End Of Test				

Test Recording Log – OE_TC_063					
Step 1 (*PushConsumer* files)	Step 2 (*push* & *disconnect_push_consumer*)	Step 3 (*PushSupplier* files)	Step 4 (*disconnect_push_supplier*)	Step 5 (*push* invocations)	Step 6 (exception handling)

Test Summary - OE_TC_063

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

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

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

OE0063_____

OE0064_____

OE0065_____

Failed Items (Section/Step Number):

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.9. OE_TC_070 - Domain Profile

Test Case Number: OE_TC_070

DomainProfile

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0588	Domain Profile files shall be complaint to the Document Type Definitions (DTDs) provided in Appendix D.
OE0590	All XML files shall have as the first two lines as an XML declaration (?xml) and a document type declaration (!DOCTYPE).
OE0591	A *Software Package Descriptor* file shall have a "*.spd.xml*" extension.
OE0592	A *Software Component Descriptor* file shall have a "*.scd.xml*" extension.
OE0594	A *Properties* File shall have a "*.prf.xml*" extension.
OE0595	A *Device Package Descriptor* File shall have a "*.dpd.xml*" extension.
OE0596	A *Device Configuration Descriptor* file shall have a "*.dcd.xml*" extension.
OE0597	A *DomainManager Configuration Descriptor* file shall have a "*.dmd.xml*" extension.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Section 3.1.3.5
SCA Appendix D: Domain Profile Attachment 1 Document Type Definitions	Version 2.2.2 15 May 2006	

Test Objective

This test case verifies OE0588, OE0590, OE0591, OE0592, OE0594, OE0595, OE0596, and OE0597. The objective of this test is to verify that the Domain Profile is compliant to the Document Type Definitions (DTD) provided in Appendix D. The tests also verify 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.

Note: If testing an SCA Extension compliant OE this test case should be replaced by OE_TC_185 as the Domain Profile test case.

Places to Verify

Device, DeviceManager, and Domain Manager XML (Domain Profile)

IDL References

None

Preconditions

- All the Domain Profile files are available

Test Description

The following XML 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 Package Descriptor	spd.xml	softpkg.2.2.2.dtd
Software Component Descriptor	scd.xml	softwarecomponent.2.2.2.dtd
Properties File	spd.xml	profile.2.2.2.dtd
Device Package Descriptor	dpd.xml	devicepkg.2.2.2.dtd
Device Configuration Descriptor	dcd.xml	deviceconfiguration.2.2.2.dtd
Domain Manager Configuration Descriptor	dmd.xml	domainmanagerconfiguration.2.2.2.dtd

A. Identify the Domain Profile files and their type used in the OE under test and verify that there is a corresponding DTD file using the above table (OE0588).
   1. **Fail**: Each Domain Profile file type does not 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 (OE0588).
   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 the *Domain Profile* file has the first two lines with *XML* declaration (?xml) and a *document type declaration* (!DOCTYPE) (OE0590).
   1. **Fail**: The *Domain Profile* file does not have the *XML or DTD* type declarations.
   2. **Pass**: The *Domain Profile* file has the *XML and DTD* type declarations.

D. Verify that each Domain Profile file has the correct file name extension as indicated in the table above (OE0591, OE0592, OE0594, OE0595, OE0596, OE0597).
   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 recording of data.

OE_TC_070				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the Domain Profile files and their type used in the OE under test and verify that there is a corresponding DTD file using the above table (OE0588).				
1. Obtain and record the directory where the Domain Profile (XML) files are located.	**Pass**: One or more directories are located. (OE0588) **Untested:** A directory is not located. (OE0588)		Multiple directories may exist.	
2. Identify and record the type of each Domain Profile (XML) file used in the OE. The types are DMD, DCD, DPD, SPD, SCD, PRF.	Required types: xxx.dmd.xml (DMD) xxx.spd.xml (SPD) Optional types: xxx.dcd.xml (DCD) xxx.dpd.xml (DPD) xxx.scd.xml (SCD) xxx.prf.xml (PRF) **Pass:** DMD and SPD files found. (OE0588) **Fail:** No DMD or SPD files found. (OE0588)		This can be done by examining the XML file extensions as indicated in the expected results column.	
3. Locate and verify the corresponding DTD file from the OE directory for each Domain Profile file type identified in Step 2.	**Pass:** Each type of Domain Profile file has a corresponding DTD file. (OE0588) **Fail:** Not each type of Domain Profile file has a corresponding DTD file. (OE0588)		See table in Test Description for File Name Extension and Matching DTD files.	
B. Verify that the *Domain Profile* files are compliant to the applicable SCA DTD (OE0588).				

OE_TC_070				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Compare *Domain Profile* files from the OE to the format of the SCA v2.2.2 DTD files.	**Pass**: The *Domain Profile* file(s) match(s) the format of the DTD files specified in the SCA. (OE0588) **Fail:** The *Domain Profile* file(s) do not match(s) the DTD files specified in the SCA. (OE0588)		A file comparison tool such as the diff or Cygwin functions in UNIX/Linux or csdiff with windows can be used for this test.	
C. Verify that the *Domain Profile* file has the first two lines with *XML* declaration (?xml) and a *document type declaration* (!DOCTYPE) (OE0590).				
5. Confirm that each Domain Profile (XML) file has the *?XML* declaration in the 1st line and *!DOCTYPE* declaration in the 2nd line.	**Pass:** The *Domain Profile* file has the *?XML* and *!DOCTYPE* declarations. (OE0590) **Fail:** The *Domain Profile* file does not have the ?*XML* or *!DOCTYPE* declarations. **(**OE0590)		Sample declaration: <?xml version="1.0" …> <!DOCTYPE…>	
D. Verify that each Domain Profile file has the correct file name extension as indicated in the table above (OE0591, OE0592, OE0594, OE0595, OE0596, and OE0597).				
6. Confirm that each *Software Package Descriptor* (SPD) file has a "*.spd.xml*" extension.	**Pass:** Each *Software Package Descriptor* (SPD) file has a "*.spd.xml*" extension. (OE0591) **Fail:** Not each *Software Package Descriptor* (SPD) file has a "*.spd.xml*" extension. (OE0591)			

OE_TC_070				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Confirm that each *Software Component Descriptor* (SCD) file has a "*.scd.xml*" extension.	**Pass:** Each *Software Component Descriptor* (SCD) file has a "*.scd.xml*" extension. **(**OE0592) **Fail:** Not each *Software Component Descriptor* (SCD) file has a "*.scd.xml*" extension. **(**OE0592)			
8. Confirm that each *Properties Descriptor* (PRF) file has a "*.prf.xml*" extension.	**Pass:** Each *Properties Descriptor* (PRF) file has a "*.prf.xml*" extension. **(**OE0594) **Fail:** Not each *Properties Descriptor* (PRF) files have a "*.prf.xml*" extension. **(**OE0594)			
9. Confirm that each *Device Package Descriptor* (DPD) file has a ".dpd.*xml*" extension.	**Pass:** Each *Device Package Descriptor* (DPD) file has a ".dpd.*xml*" extension. **(**OE0595) **Fail:** Not each *Device Package Descriptor* (DPD) file has a ".dpd.*xml*" extension. **(**OE0595)			
10. Confirm that each *Device Configuration Descriptor* (DCD) file has a ".dcd.*xml*" extension.	**Pass:** Each *Device Configuration Descriptor* (DCD) file has a ".dcd.*xml*" extension. **(**OE0596) **Fail:** Not each *Device Configuration Descriptor* (DCD) file has a ".dcd.*xml*" extension. **(**OE0596)			

OE_TC_070				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
11. Confirm that each *DomainManager Configuration Descriptor* (DMD) file has a ".dmd.*xml*" extension.	**Pass:** Each *DomainManager Configuration Descriptor* (DMD) file has a ".dmd.*xml*" extension. **Fail:** Not each *DomainManager Configuration Descriptor* (DMD) file has a ".dmd.*xml*" extension. (OE0597)			
End Of Test				

Test Recording Log - OE_TC_070					
Step1 (directory containing the Domain Profile files)					
Step2 (Domain Profile file type)	**Step3** (DTD file)	**Step4** (Compliant to SCA DTD)	**Step5** (?XML and !DOCTYPE declarations)	**Step6** ("*.spd.xml*" extension.)	
Step7 ("*.scd.xml*" extension)	**Step8** (".prf.*xml*" extension)	**Step9** (".dpd.*xml*" extension)	**Step10** (".dcd.*xml*" extension)	**Step11** (".dmd.*xml*" extension)	

Test Summary OE_TC_070

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

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

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

OE0588_____

OE0590_____

OE0591_____

OE0592_____

OE0594_____

OE0595_____

OE0596_____

OE0597_____

Failed Items (Section/Step Number):

_____
_____
_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.5.10. OE_TC_075 - Provided interfaces described as provides ports

Test Case Number: OE_TC_075

Application Interfaces - Provided interfaces described as *provides* ports

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0614	Interfaces provided by a component shall be described in a Software Component Descriptor file as *provides* ports.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-96 Section 3.2.2
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Pages D-40 - D-45, Section D.6.1.5.1

Test Objective

This test case verifies OE0614. The objective of this test is to verify that interfaces provided by a component are described the component's Software Component Description (SCD) file as *provides* ports.

Places to Verify

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

IDL References

None

Preconditions

- A successful run of the test case OE_TC_119, a test of the Domain Profile Files, is required., and a listing (or diagram) of all *provides* ports by component for all components is available from the test results. The *provides* ports information can only be obtained from the SCD files.
- The developer's Interface Control Document (ICD) or other interface documentation of the Core Framework(CF) are available.

Test Description

Note: The list of components mentioned in the Preconditions is a result of a semi-automated test case which is verifying the Domain Profile files. The SCD files are a part of the Domain Profile.

A. Identify each component which provides interfaces based on the developer's supplied ICD. (OE0614)
   1. **Untested**: No ICD or other interface documentation is provided by the developer.
   2. **N/A:** No components are found which provide interfaces.

For each component in the ICD which provides interfaces perform the following step:

B. Verify that the identified interfaces are described in the component's SCD (from OE_TC_119 results). (OE0614)
   1. **Pass:** The *provides* ports of the SCD match those described as provided interfaces in the ICD.
   2. **Fail:** The *provides* ports of the SCD does not match those described as provided interfaces in the ICD.

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 OE_TC_119 is the provides ports of the OE.

OE_TC_075				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify each component which provides interfaces based on the developer's supplied ICD. (OE0614)				
1. Identify each component which provides interfaces based on the developer's supplied ICD	**Untested**: No ICD or other interface documentation is provided by the developer. (OE0614) **N/A**: No components are found which provide interfaces. There is no way to verify entries found in the SCD. (OE0614)			
For each component in the ICD which provides interfaces perform the following step:				
B. Verify that the identified interfaces are described in the component's SCD (from OE_TC_119 results). (OE0614)				
2. Identify and list the provided interfaces for the component as described by the ICD.	A list of ICD described provided interfaces.			
3. Find the corresponding component in the list (or diagram) of *provides* ports generated from the OE_TC_119 test case.	**Pass:** A corresponding component is found in the list (or diagram). (OE0614) **Fail:** A corresponding component is not found in the list (or diagram). (OE0614)		A corresponding component would be one with the same name or a very similar name. Should an undocumented component be found in the OE_TC_119 results, please report it to the developer and make a note that the ICD is incomplete and why.	

OE_TC_075				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Identify and list the *provides* ports for the corresponding component found in Step 3. This should come from the list (or diagram) generated from the OE_TC_119 test case.	**Pass**: A list of *provides* port(s) is found. (OE0614) **Fail:** No list of *provides* port(s) is found. (OE0614)		Should an undocumented provides port be found in the OE_TC_119 results, please report it to the developer and make a note that the ICD is incomplete and why.	
5. Match the list created in Step 2 with the list created in Step 4.	**Pass:** The list of provided interfaces and *provides* ports match. (OE0614) **Fail:** The list of provided interfaces and *provides* ports do not match. (OE0614)			
End Of Test				

Test Recording Log – OE_TC_075				
Step 1 (Components which provide interfaces per the ICD)	**Step 2** (A component's provided interfaces)	**Step 3** (Corresponding component in the Domain Profile list (from OE_TC_119))	**Step 4** (Corresponding component's list of *provides* ports)	**Step 5** (Interfaces listed in Step 2 and provides ports listed in Step 4 match?) Y/N

Test Summary OE_TC_075

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

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

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

OE0614_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.11. OE_TC_076 - Required interfaces described as uses ports

Test Case Number: OE_TC_076

Application Interfaces - Required interfaces described as *uses* ports

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0615	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)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-96 Section 3.2.2
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Pages D-40 - D-45, Section D.6.1.5.1

Test Objective

This test case verifies OE0615. The objective of this test is to verify that interfaces required by a component are described in the component's Software Component Description (SCD) file as *uses* ports.

Places to Verify

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

IDL References

None

Preconditions

- A helpful but not required precondition is a successful run of the test case OE_TC_119, a test of the Domain Profile Files. From it one can use a listing (or diagram) of the *uses* ports by component for all components. The *uses* ports information can only be obtained from the SCD files.
- The developer's Interface Control Document (ICD) or other interface documentations of the Core Framework(CF) are available.

Test Description

Note: The list of components mentioned in the Preconditions is a result of a semi-automated test case which is verifying the Domain Profile files. The DCD and SCD files are a part of the Domain Profile.

A. Identify each component which requires interfaces based on the developer's supplied interface documentation. (OE0615)
   1. **Untested**: No interface documentation is provided by the developer.
   2. **N/A:** No components are found which require interfaces.

For each component in the interface documentation which requires interfaces perform the following step:

B. Verify that the identified interfaces are described in the devices DCD or the components SCD (from OE_TC_119 results or other XML review). (OE0615)
   1. **Pass:** The *uses* ports of the DCD or SCD match those described as required interfaces in the interface documentation.
   2. **Fail:** The *uses* ports of the DCD or SCD does not match those described as required interfaces in the interface documentation.

Manual Test Steps

Notes: 1. The information from OE_TC_119 is the uses ports of the OE.
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 recording of data.

OE_TC_076				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify each component which requires interfaces based on the developer's supplied interface documentation. (OE0615)				
1. Identify each component which requires interfaces based on the developer's supplied interface documentation.	**Untested**: No interface documentation is provided by the developer. (OE0615) **N/A**: No components are found which require interfaces. There is no way to verify entries found in the SCD. (OE0615)			
2. Identify and list the required interfaces for the component as described by the interface documentation.	A list of interface documentation described required interfaces.			
3. Locate and list the source code for the components that require interfaces as described by the interface documentation Use the list from **Step 2** as needed..	A list of components' source code that use interfaces.		The documentation can help find the source code. One can also ask the developer what source code has *uses* ports.	
For each component in the interface documentation which requires interfaces perform the following step:				
B. Verify that the identified interfaces are described in the component's DCD or SCD file. (from OE_TC_119 results or other XML review). (OE0615)				
If the results of test OE_TC_119 are available for uses ports you may skip to step 6.				

OE_TC_076				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Review the DCD file within the *devicemanagersoftpkg* element to identify its SPD files. Use the *descriptor* element in the SPD files to identify any SCD files	Make a list of the SCD files related to this OE.			
5. Review the SCD files within the *componentfeature* element for uses ports	Make a list of the uses ports by component.			
6. Find the corresponding component in the list from **Step 5**, that is from the SCD (or diagram generated from the OE_TC_119 test case), of *uses* ports.	**Pass:** A corresponding component is found in the list (or diagram). (OE0615) **Fail:** A corresponding component is not found in the list (or diagram). (OE0615)		A corresponding component would be one with the same or very similar names. Should an undocumented component be found, please report it to the developer and make a note that the interface documentation is incomplete and why.	
7. Identify and list the *uses* ports for the corresponding component found in **Step 6**. This should come from the list create in Step 5 (or diagram generated from the OE_TC_119 test case).	**Pass:** A list of *uses* port(s) is found. (OE0615) **Fail:** No list of *uses* port(s) is found. (OE0615)		Should an undocumented provides port be found, please report it to the developer and make a note that the interface documentation is incomplete and why.	
8. Match the list created in **Step 3** with the list created in **Step 7**.	**Pass:** The list of required interfaces and *uses* ports match. (OE0615) **Fail:** The list of required interfaces and *uses* ports do not match. (OE0615)			
End Of Test				

Test Recording Log – OE_TC_076				
Step 2 (documented required interfaces)	**Step 3** (Component source code that requires interfaces)	**Step 6** (*uses* ports from the SCD files)	**Step 8** (Component source code required interfaces & SCD *uses* ports match – Y/N?)	**Notes**

Test Summary OE_TC_076

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

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

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

OE0615_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.12. OE_TC_077 - SCA APIs and non-IDL interfaces

Test Case Number: OE_TC_077

OE Interfaces

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0741	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.
OE0742	All SCA APIs shall have their interfaces described in IDL.
OE0743	All non-IDL interfaces shall provide an IDL mapping within the service definition.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-96, Section 3.2.2.1
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Pages D-28 – D-31, Section D.5.1
SCA, Appendix C: Core Framework IDL	V2.2.2 15 May 2006	All
The Common Object Request Broker: Architecture and Specification, Object Management Group, Inc (OMG)	V 3.0 July 2002	Chapter 3 OMG IDL Pages 3-2; 3-20 thru 3-26

Test Objective

This test case verifies OE0741, OE0742 and OE0743. The objective of this test is to verify that another engineering group can address any interface and data exchange issues within and between operating environment and applications during a later maintenance software cycle. The test depends upon the usability of the available Interface Control Documents (ICD) for the non-standard interfaces. Therefore, all APIs implemented in a JTR set must have their interfaces generated through the use of an IDL. Additionally, all IDL interfaces that are not defined by SCA documents (SCA v2.2.2, APIs) need be accurately and thoroughly described using a formal service definition in the ICDs of the Core Framework. This test case also checks that APIs generated using methods other than IDL must have an IDL mapping in the service definition.

The SCA definition of service definitions is: “SCA service definitions consist of APIs, behavior, state, priority and additional information that provide the contract between the Service Provider and the Service User.”

Standard vs. Non-standard Interfaces

The criteria of a 'standard' interface (and thereby, inferring a 'non-standard' interface) is an interface that is equivalent to 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 (OE0741): 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.

Places to Verify

Domain Profile files, and Core Framework documentation

IDL References

None

Preconditions

- Have a clear understanding/agreement between the developer/vendor and the tester of what is the definition of a standard interface and what is a non-standard interface. (Keep in mind that there are no known complete official lists of government-approved standards or commercially-accepted standards.) The developer/vendor will provide the burden of proof with convincing documentation describing the non-standard interface descriptions with quality, quantity and completeness.
- ICD document must be available.
- The Core Framework (including devices and services) documentation is available.
- The Domain Profile files of the Core Framework are available.
- The IDL files of the Core Framework (including devices and services) are available.

Test Description

For each DCD file of the OE under test:

A. Identify all the APIs declared in the Domain Profile. (OE0741, OE0742, OE0743)
   1. **Pass**: APIs are declared in the Domain Profile files.
   2. **Fail**: No APIs are declared in the Domain Profile files.
B. Verify that all APIs implemented in a JTR set must have their interfaces generated through the use of an IDL. (OE0742)

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. (OE0741)
   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. (OE0741)
   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. (OE0741)
   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 'not enough' non-standard interface information that has been documented and available. This completeness level can be somewhat subjective, but 'enough' means that the scope of coverage and level of detail exists for another developer to continue the software development for the interfacing.

F. For each API that is not generated from an IDL verify that its interface is mapped in the service definition in an IDL format. (OE0743)
   1. **Pass**: All non-IDL generated APIs are mapped in an IDL format.
   2. **Fail**: Not all non-IDL generated APIs are mapped in an IDL format.

Manual Test Steps

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

OE_TC_077				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Results**
For each DCD file of the OE under test:				
A. Identify all the APIs declared in the Domain Profile. (OE0741, OE0742, OE0743)				
1. Locate and open the DCD file.	**Pass:** The DCD file is located. (OE0741, OE0742, OE0743) **Untested:** There is no DCD file. (OE0741, OE0742, OE0743)			
2. Identify and record all Software Package Descriptor (SPD) declarations in the DCD file.	**Pass:** The SPD declarations are recorded. (OE0741, OE0742, OE0743) **Fail:** There are no SPD declarations found. (OE0741, OE0742, OE0743)		Sample declaration: <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.	**Pass:** The SCD declarations are recorded. (OE0741, OE0742, OE0743) **Fail:** There are no SCD declarations found. (OE0741, OE0742, OE0743)		Sample declaration: <descriptor> <localfile name= "**xxxxx.scd.xml**" />	
4. Identify and record all the *supportsinterface* declarations from the each of the SCDs declared in Step 3.	**Pass:** The *supportsinterface* declarations are recorded. (OE0741, OE0742, OE0743) **Untested:** There are no *supportsinterface* declarations. (OE0741, OE0742, OE0743)		Sample declaration: <componentfeatures> <**supportsinterface** repid= "**IDL:CF/yyyyy**" **supportsname**="**yyyyy**" />	

OE_TC_077				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Results**
5. Identify all the *uses* interfaces declarations from the each of the SCDs declared in Step 3.	**Pass:** The *uses* interface declarations are recorded. (OE0741, OE0742, OE0743) **Untested:** There are no *uses* interface declarations. (OE0741, OE0742, OE0743)		Sample declaration: <ports> <**uses** repid="**IDL:CF/zzzzz**" **usesname**="**zzzzz**">	
6. Identify and record all the *provides* interfaces declarations from the each of the SCDs declared in Step 3.	**Pass:** The *provides* interface declarations are recorded. (OE0741, OE0742, OE0743) **Untested:** There are no *provides* interface declarations. (OE0741, OE0742, OE0743)		Sample declaration: <ports> <**provides** repid= "**IDL:nnnnn**" **providesname**="**nnnnn**">	
B. Verify that all APIs implemented in a JTR set must have their interfaces generated through the use of an IDL. (OE0742)				

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.
 	
Record the IDL generated interfaces and the non-IDL generated interfaces.
	Pass: Each interface declared is described and generated through the use of an IDL file.
	Interfaces are recorded. (OE0742)
	
	Fail: Each interface declared is not described and generated through the use of an IDL file.
	Interfaces are recorded. (OE0742)
			interface Aaaaa : Bbbbb {
			struct AaaBbbType {
			string CccID;
			unsigned long Ddddd;
			};
			interface GgggHhhh {
			exception UnknownGggg
			{ };	

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. (OE0741)				
8. Locate and write down the name of all resource components declared in the DCD file in the Log Sheet.			Interface declarations are in the SCD files, so review the DCD which points to the SPD which then points to the SCD. (Note: bolded part of	

OE_TC_077				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Results**
Resources are declared in the <componentfile> element of the DCD file. These elements are optional.			name is vendor-specific example.) **DCD examination:** Locating the componentfile and SPD file declarations in the DCD file. <componentfile id="***BlackComponent_File***" > <localfile name="***BlackComponent.spd.xml***"/> </componentfile> **SPD Examination**: From the SPD, locate and record the SCD propertyfile declaration. </propertyfile> <descriptor> <localfile name="***BlackComponent.scd.xml***"/> </descriptor> **SCD Examination:** From the SCD, locate and record all the interfaces declarations. <interfaces> <interface repid="IDL:CF/Resource:1.0" name="Resource"> **interfaces** <inheritsinterface repid="IDL:***CF/PortSupplier***:1.0"/> … <interfaces>	
9. Examine all vendor-provided interface documents for software	Documents are examined for subsequent test steps.		Application interfaces focus on software. Hardware interfaces addressed in the	

OE_TC_077				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Results**
interface descriptions and tag/identify the locations for the next test step	**Untested**: If unable to locate interface documents. (OE0741)		Operating System (OE) test case, tentatively identified as OE_TC_163. Interfaces within and between applications to be considered are: 1. Base Application interfaces (such as, Port, Port Supplier, Resource, ResourceFactory, LifeCycle, Testable Object, and PropertySet interfaces) 2. Framework Control interfaces (such as Device interfaces and Domain interfaces) 3. Framework Services File interfaces (such as File, FileSystem and FileManager interfaces) Note: An example of the interface document would be the Interface Control Documents which would include specifications, diagrams and other interface descriptions.	
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. (OE0741)				

OE_TC_077				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Results**
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. If only standard interfaces are discovered, then skip to the End of the Test.	**Untested**: (May or may not find non-standard interfaces.) (OE0741)		Note: The following criteria of a 'standard' interface (and thereby, infers a 'non-standard' interface) is an interface that is equivalent to 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.	
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 in any other documents. (OE0741)				

OE_TC_077				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Results**
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.	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. **Pass:** The ICD contain information to properly describe the non-standard interfaces. (OE0741) **Fail:** The ICD does not contain "enough" information to properly describe the non-standard interfaces. (OE0741)		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'.	
F. 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. (OE0743)				
12. Examine each interface in Step 4 to 6 that is non-IDL generated and verify that it is documented service definition in an IDL format.	**Pass:** Each non-IDL generated API is documented following the service definition. (OE0743) **Fail:** Each non-IDL generated API is not documented following the service definition. (OE0743)		These APIs should be documented similar to the API descriptions in Appendix C of SCA 222. Some possible methods of providing this detail are with a header file or an Interface Control Document.	
End Of Test				

Test Recording Log – OE_TC_077				
DCD File				
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 OE_TC_077

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

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

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

OE0741_____

OE0742_____

OE0743_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.13. OE_TC_099 - General Rules : Higher Order Language

Test Case Number: OE_TC_099

General Rules : Higher Order Language

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0628	Software developed for an SCA-compliant system shall be developed in a standard higher order language.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Section 3.4.1.1

Test Objective

This test case verifies OE0628. The objective of this test is to verify through inspection that new OE Software developed for an SCA-compliant system is developed 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

All OE source code that is available that is to be certified.

IDL References

None.

Preconditions

- All OE source code that is to be certified should be available. A source file list, such as those found in the Version Description Document (VDD), might help to identify the needed OE source code. Proprietary outside vendor source code files, however, may be unobtainable, and would not then be certifiable.

Test Description

For each source code file in the OE,

A. Verify manually that the non-legacy code has been developed with a standard higher order language as described in the Test Objective section (OE0628).
   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. Test Result will include Pass, Fail, Untested, or N/A.
2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_099				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify manually that the non-legacy code has been developed with a standard higher order language as described in the Test Objective section. (OE0628)				
1. Identify and record which OE code is 'non-legacy' source code.	**Pass**: Can identify 'non-legacy' source code. (OE0628) **Fail**: Cannot identify 'non-legacy' source code. (OE0628)		Note: 'non-legacy' source code is defined per JTNC Standards as "software components of the radio that were produced before the software code development for SCA compliancy was started."	
2. Determine that the operating environment source code within the SCA boundary was developed in standard higher order language.	**Pass**: The operating environment source code within the SCA boundary was developed in standard higher order language. (OE0628) **Fail**: The operating environment source code within the SCA boundary was developed in standard higher order language. (OE0628)		Examples of what are <u>not</u> a standard higher order language are Assembly languages and VHDL.	
End Of Test				

Test Recording Log – OE_TC_099	
Step 1 (non-legacy code, such as name of files, function, procedures, etc.)	Step 2 (What Higher Order Language was used)

Test Summary OE_TC_099

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

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

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

OE0628_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.14. OE_TC_100 - Legacy Software interfaces

Test Case Number: OE_TC_100

Legacy Software Interfaces

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0630	Legacy software shall interface with the Core Framework in accordance with this specification.

References

Document Name	Version/Date	Location (Pages, Section)
SCA Specification	Version 2.2.2 15 May 2006	Page 3-98, Section 3.4.1.2 Pages 3-5 to 3-94, Section 3.1.3
Support and Rationale Document (SRD) for the SCA	V2.2 19 Dec 2001	Page 3-31
Wikipedia	24 July 2008	http://en.wikipedia.org/wiki/Wrapper_pattern

Test Objective

This test case verifies OE0630. The objective of this test is to verify that legacy software components of the Operating Environment that interface with the Core Framework are SCA compliant. Even though the term "legacy" is not defined in the SCA document, the generally accepted definition of the term is software components of the radio that were produced before the OE's development for SCA compliancy began.

Places to Verify

Legacy software interfaces and their documentation.

IDL References

None

Preconditions

- All of the source code files for the OE and legacy software under test are available.

- SCA related software design documentation is 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 OE's SCA architecture.

Test Description

A. Determine if the OE contains any legacy software. (OE0630)
   1. **N/A**: No legacy software exists in the OE.

For each component of legacy software:

B. From the OE design documentation that is provided by the developer, identify the component(s) of the SCA architecture, with which the legacy software integrates or interfaces with on the OE. In most cases this is going to be a port or device. (OE0630)
   1. **N/A:** The legacy software is not interfacing with the OE.

For each component of legacy software that integrates or interfaces with a component of the SCA architecture for the OE:

C. Verify that all function calls made from the legacy software to the CF Interfaces either are SCA compliant or go through a wrapper, which is SCA compliant. This specifically covers interfaces the legacy software **requires**.
   1. **Pass**: The legacy software of the OE and its documentation identifies, documents and follows the CF and therefore, interfaces with CF according to the SCA.
   2. **Fail**: The legacy software is being used in the OE but cannot be identified with a part of the OE's SCA architecture. The component does not follow the CF as described in the SCA v2.2.2, Section 3.1.3

D. Verify that all function calls made from the CF to the legacy software either are SCA compliant or go through a wrapper, which is SCA compliant. This specifically covers interfaces the legacy software **provides**.
   1. **Pass**: The legacy software of the OE and its documentation identifies and documents how the CF calls the legacy software and therefore, the CF interfaces with the legacy software according to the SCA.
   2. **Fail**: The legacy software is being used in the OE but cannot be identified with a part of the OE's SCA architecture. The component does not follow the CF as described in the SCA v2.2.2, Section 3.1.3

Note: Wrapper as it is used here is defined at Wikipedia (http://en.wikipedia.org/wiki/Wrapper_pattern) as the following:

> In computer programming, the adapter design pattern (often referred to as the wrapper pattern or simply a wrapper) translates one interface for a class into a compatible interface.

Manual Test Steps

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

OE_TC_100				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Determine if the OE contains any legacy software. (OE0630).				
1 Determine if the source code contains any components that were developed before the waveform/OE began SCA development. This includes such items as FPGA and DSP images.	The developer must provide an accurate list of all legacy software included in the SDD and other interface documents. Record that list. **N/A**: No legacy software exists in the OE. (OE0630)			
For each component of legacy software:				
B. From the OE design documentation that is provided by the developer, identify the component(s) of the SCA architecture, with which the legacy software integrates or interfaces with on the OE. In most cases this is going to be a port or device. (OE0630).				
2. Identify the component(s) of the SCA architecture on the OE with which the legacy component is identified or integrated.	The legacy software must interface or integrate with component(s) of the SCA architecture. **N/A:** The legacy software exists in the OE but is not interfacing with the OE. (OE0630)		The component(s) should be identified as part of the radio's OE (application is a separate test). More specifically it must be a port, device or other SCA architectural component.	
For each component of legacy software that integrates or interfaces with a component of the SCA architecture for the OE:				
C. Verify that all function calls made from the legacy software to the CF either are SCA compliant or go through a wrapper, which is SCA compliant. This specifically covers interfaces the legacy software requires. (OE0630).				

OE_TC_100				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that all function calls made from the legacy software to the CF either are SCA compliant or go through a wrapper, which is SCA compliant.	**Pass**: The legacy software of the OE identifies, documents and follows the CF and which, therefore, interfaces with Core Framework according to the SCA. (OE0630) **Fail**: The legacy software is being used in the OE but cannot be identified with a part of the OE's SCA architecture. The component does not follow the CF Interfaces as described in the SCA v2.2.2, Section 3.1.3. (OE0630)		For each of the next steps one may be looking for "wrapper" code for the specified interfaces that are needed. Components to consider for these next test steps are GPP, FPGA and DSP images. SCA compliant means that it complies with the SCA Specification version 2.2.2 mentioned in the Reference table above. Identifies means that the documentation that addresses the legacy software identifies the components of which the legacy software is a part. Documents: This means that the documentation that addresses the legacy software describes the interfaces to and from it. Follows: This means that the software as implemented conforms to the interfaces described in the documentation that addresses the legacy software.	
D. Verify that all function calls made from the CF to the legacy software either are SCA compliant or go through a wrapper, which is SCA compliant. This specifically covers interfaces the legacy software provides. (OE0630).				

OE_TC_100				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that all function calls made from the CF Interfaces to the legacy software either are SCA compliant or go through a wrapper, which is SCA compliant.	**Pass**: The legacy software of the OE identifies and documents how the CF calls the legacy software and which, therefore, the Core Framework interfaces with the legacy software according to the SCA. (OE0630) **Fail**: The legacy software is being used in the OE but cannot be identified with a part of the OE's SCA architecture. The component does not follow the CF Interfaces as described in the SCA v2.2.2, Section 3.1.3 (OE0630)			
End Of Test				

Test Recording Log – OE_TC_100			
Step 1	**Step 2**	**Step 3**	**Step 4**
List of legacy software that interfaces with the OE.	Page in document & List of SCA architectural components that interface/integrate with legacy software	Does the legacy software adhere to the specifications in the given interface of the SCA v2.2.2?	
Software Design Document name		legacy software TO Core Framework (Y/N)	Core Framework TO legacy software (Y/N)

Test Recording Log – OE_TC_100			
Step 1	**Step 2**	**Step 3**	**Step 4**
List of legacy software that interfaces with the OE.	Page in document & List of SCA architectural components that interface/integrate with legacy software	Does the legacy software adhere to the specifications in the given interface of the SCA v2.2.2?	

Test Summary OE_TC_100

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

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

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

OE0630_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.15. OE_TC_101 - Hardware Critical Interfaces

Test Case Number: OE_TC_101

Device::Logical Device

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0633	Hardware critical interfaces shall be defined in Interface Control Documents that are available to other parties without restriction.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-58 Section 3.1.3.3.1.1;Page 3-98, Section 3.3.1
SCA Appendix A: Glossary	Version 2.2.2 15 May 2006	Page 88

Test Objective

This test case verifies OE0633. The objective of this test is to verify that all hardware critical interfaces are defined in Interface Control Documents (OE0633). The definition of hardware critical interfaces is described to be those interfaces at the physical boundary of a replaceable device that are required for the operation and maintenance of the device.

Places to Verify

Logical Devices/Interface Documentation which includes hardware

IDL References

None

Preconditions

- Operating Environment(OE) Documentation is available.

Test Description

A. Identify the hardware critical interfaces that are boundaries of the physical device and are required for the maintenance of the device. (OE0633)
   1. **Pass:** The hardware critical interfaces are identified.

2. **N/A**: There are no hardware critical interfaces identified for the device.

B. Verify that the hardware critical interfaces are defined in Interface Control Documents (ICD). (OE0633)
1. **Pass**: The hardware critical interfaces are defined in ICDs.
2. **Fail**: The hardware critical interfaces are not defined in ICDs.
3. **N/A**: There are no hardware critical interfaces identified in the Operating Environment.

NOTE: Application Program Interface is terminology used interchangeably with ICD.

Manual Test Steps

Notes: 1. Test Result will include Pass, Fail, Untested, or N/A.
2. The Test Recording Log sheet is intended to record data for each step that requires recording of data.

OE_TC_101				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the hardware critical interfaces that are boundaries of the physical device and are required for the operation and maintenance of the device. (OE0633)				
1. Identify the hardware critical interfaces that are boundaries of the physical device and required for the operation and maintenance of the device.	**NA**: The hardware critical interfaces cannot be identified. (OE0633)		SCA v2.2.2, Page 3-98. "Critical interfaces are those interfaces at the physical boundary of a replaceable device that are required for the operation and maintenance of the device."	
Verify that the hardware critical interfaces are defined in Interface Control Documents (ICD). (OE0633) **NOTE: Application Program Interface is terminology used interchangeably with ICD.**				
2. Verify that the hardware critical interfaces are defined in the ICD. List the ICD name.	1. **Pass:** The hardware critical interfaces are defined in the ICD. (OE0633) 2. **Fail:** The hardware critical interfaces are *not* defined in the ICD. (OE0633) 3. 4. **N/A:** There are *no* hardware critical interfaces identified. (OE0633)		The interfaces may actually be defined in documentation that describes the APIs of the device. ICD is an overarching term used to describe documentation for interfaces.	
End Of Test				

Test Recording Log-OE_TC_101		
Step 1 (Name of hardware critical interface for boundary area)	**Step 2** (Hardware Critical Interface Defined in ICDs) (Y/N?)	**Step 2 (cont'd)** (List ICD(s).)

Test Summary - OE_TC_101

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

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

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

OE0633_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.16. OE_TC_117 - DTD files

Test Case Number: OE_TC_117

Domain Profile

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0589	DTD files are installed in the domain and shall have ".dtd" as their filename extension.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-90 to 3-91, Section 3.1.3.5,
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	D-2
Attachment 1 of Appendix D: Domain Profile Document Type Definitions	Version 2.2.2 15 May 2006	All

Test Objective

This test case verifies requirement OE0589. The objective of this test is to verify that the DTD (Document Type Definitions) files are installed in the domain and have the "***.dtd***" filename extensions. The document type declaration (!DOCTYPE) specifies the DTD for the document. The names of the DTD files installed in the domain are found in SCA v2.2.2 Attachment 1 to Appendix D.

Places to Verify

DeviceManager, Device

Parameters

None

Preconditions

- The Domain Profile files used by the OE must be available.

Test Description

A. Identify the Domain Profile XML files in the OE.

1. **Pass:** At a minimum, the DomainManager Configuration Descriptor and Software Package Descriptor files are found.
2. **Fail:** DomainManager Configuration Descriptor and Software Package Descriptor files are not found.

For each Domain Profile XML file found, perform the following:

B. Verify that the document type declaration (!DOCTYPE) specifies a file having the extension ".dtd". (OE0589)
   1. **Pass:** The document type declaration specifies a file having the extension ".dtd".
   2. **Fail:** The document type declaration does not specify a file having the extension ".dtd".
C. Verify that each DTD (Document Type Definitions) follows the naming convention of Attachment 1 to Appendix D. (OE0589)
   1. **Pass:** All DTD files follow the naming convention.
   2. **Fail:** A DTD file does not follow the naming convention.

Manual Test Steps

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

OE_TC_117				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the Domain Profile XML files in the OE. OE0589)				
1. Identify and record the full pathname of the Domain Profile XML files.	**Pass:** At a minimum, the Domain Manager Configuration and Software Package Descriptor files are found. **(OE0589)** **Fail:** Domain Manager Configuration Descriptor and Software Package Descriptor files are not found. (**OE0589)**		Required: xxx.**dmd**.xml and xxx.**spd**.xml Optional: xxx.**dcd**.xml xxx.**dpd**.xml xxx.**scd**.xml xxx.**prf**.xml	
For each Domain Profile XML file found in Step 1, perform the following:				
B. Verify that the document type declaration (!DOCTYPE) specifies a file having the extension ".dtd". (OE0589)				
2. Verify that the document type declaration specifies the extension "**.dtd**".	**Pass:** The document type declaration specifies a file having the extension "**.dtd**". (**OE0589)** **Fail:** The document type declaration does not specify a file having the extension "**.dtd**" (**OE0589)**		This can be done by examining the XML file name extensions as indicated in the expected results column.	
C. Verify that each DTD (Document Type Definitions) follows the naming convention of Attachment 1 to Appendix D. ". (OE0589)				

OE_TC_117				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that each DTD follows the naming convention in Attachment 1 of Appendix D.	**Pass:** A DTD file follows the naming convention. **(OE0589)** **Fail:** Any DTD file not following the naming convention. **(OE0589)**		Attachment 1 of Appendix D – Domain Profile Document Type Definitions.	
End Of Test				

Test Recording Log – OE_TC_117		
Step1 (Domain Profile XML files)	**Step2** (".dtd" extension – **Y/N**)	**Step3** (naming convention – **Y/N)**

Test Summary OE_TC_117

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

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

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

OE0589 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.17. OE_TC_119 - Domain Profile files

Test Case Number: OE_TC_119

OE Domain Profile

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0613	An application, each application component, and each device manager shall be accompanied by the appropriate Domain Profile files per section 3.1.3.5.
OE0613-C146	The Software Component Descriptor (SCD) defines the CORBA interfaces supported and used by a specific component.
OE0613-C147	Within the specification of a software package descriptor several other files are referenced including a component level property file and a software component descriptor file. Within any given implementation there may be additional property files.
OE0613-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.
OE0613-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.
OE0613-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.
OE0613-C151	Duplicated properties must be the same property type, only the value can be over-ridden.
OE0613-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 componentinstantiation, componentproperties or resourcefactoryproperties element.
OE0613-C153	The propertyfile element is used to indicate the local filename of the Property Descriptor file associated with the Software Package.
OE0613-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.
OE0613-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).
OE0613-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.
OE0613-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.
OE0613-C158	The stack size and priority are options parameters used by the CF ExecutableDevice execute() operation.
OE0613-C159	Data types for the values of these options are unsigned long.

OE0613-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."
OE0613-C161	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.
OE0613-C162	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.
OE0613-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.
OE0613-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
OE0613-C165	Only simple elements can be used as execparam types.
OE0613-C166	At a minimum, the component interface has to be a CF Resource, CF ResourceFactory, or CF Device interface.
OE0613-C178	The implementation of the domain manager is itself described by the DomainManager Configuration Descriptor (DMD) which provides the location of the (SPD) file for the specific DomainManager implementation to be loaded.
OE0613-C179	The devicepkg id attribute uniquely identifies the package and is a DCE UUID, as defined in paragraph D.2.1.
OE0613-C180	The hwdeviceregistration id attribute uniquely identifies the device and is a DCE UUID, as defined in paragraph D.2.1.
OE0613-C181	The profile element is used to specify an absolute file pathname relative to a mounted CF FileSystem.

References

Document Name	Version/Date	Location (Pages, Section)
SCA	Version 2.2.2 15 May 2006	Pages 3-95, Section 3.2.1.3; Page 3-90, Section 3.1.3.5
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	All
Spectra SDR Power Tools-User Guide	Version 2.0	All
Spectra XML Validation Capability	21 January 09, By PrismTech	All

Test Objective

This test case verifies OE0613. The objective of this test is to verify that the OE 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

OE Domain Profile Files

IDL References

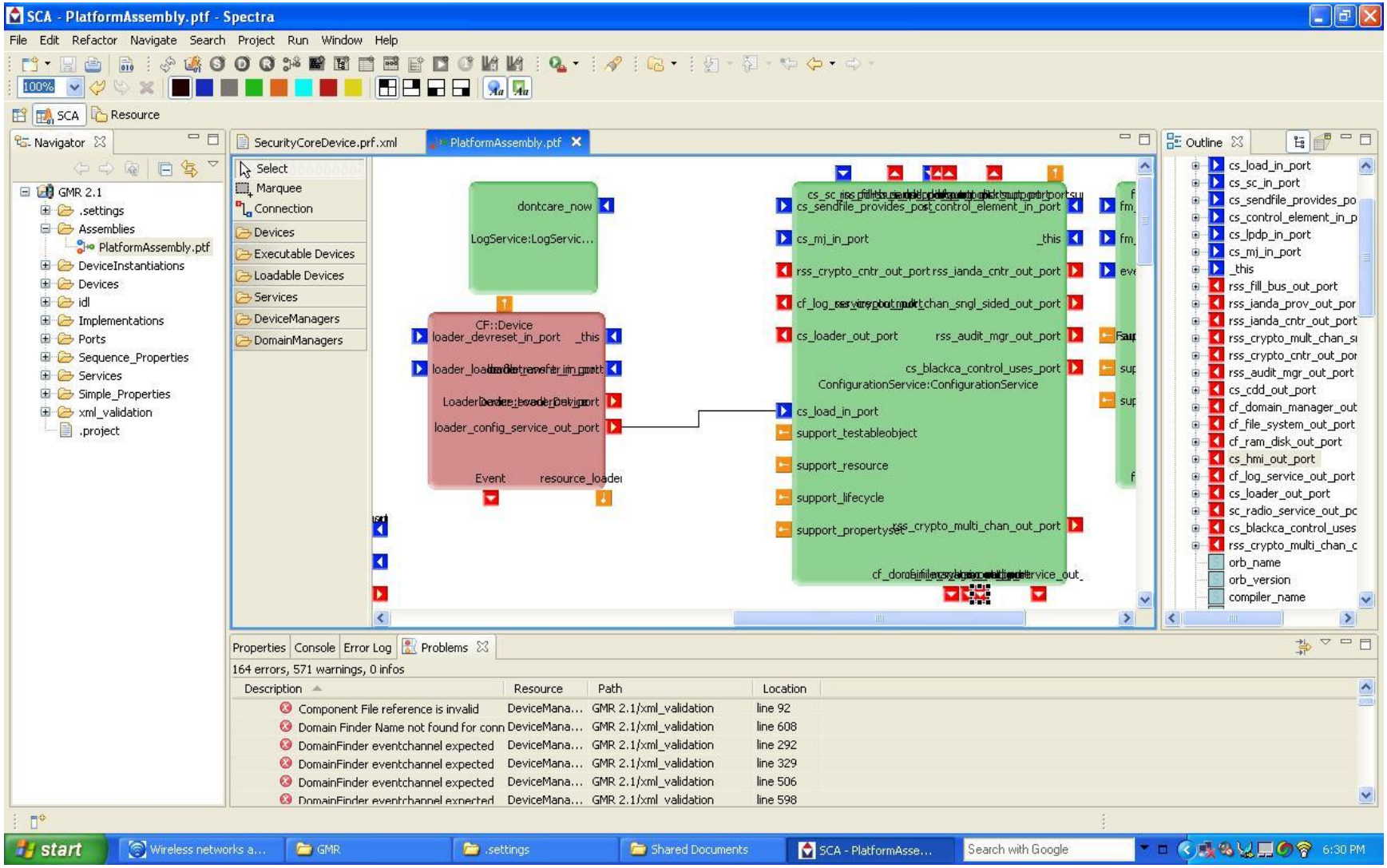
None.

Preconditions

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

Test Description

A. Set up a workspace for the OE 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.(OE0613, OE0613-C146 to OE0613-C166, OE0613-C178, OE0613-C179, OE0613-C180 and OE0613-C181)
    1. **Pass:** SCA errors are not found.
    2. **Fail:** SCA errors are found.
E. Verify that the appropriate Domain Profile files are present. (OE0613)
    1. **Pass:** No XML files are missing.
    2. **Fail:** XML files are missing.

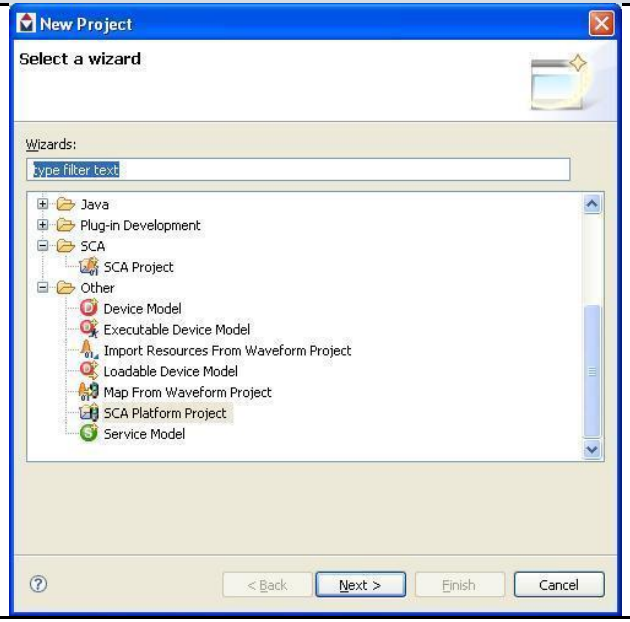


Spectra desktop

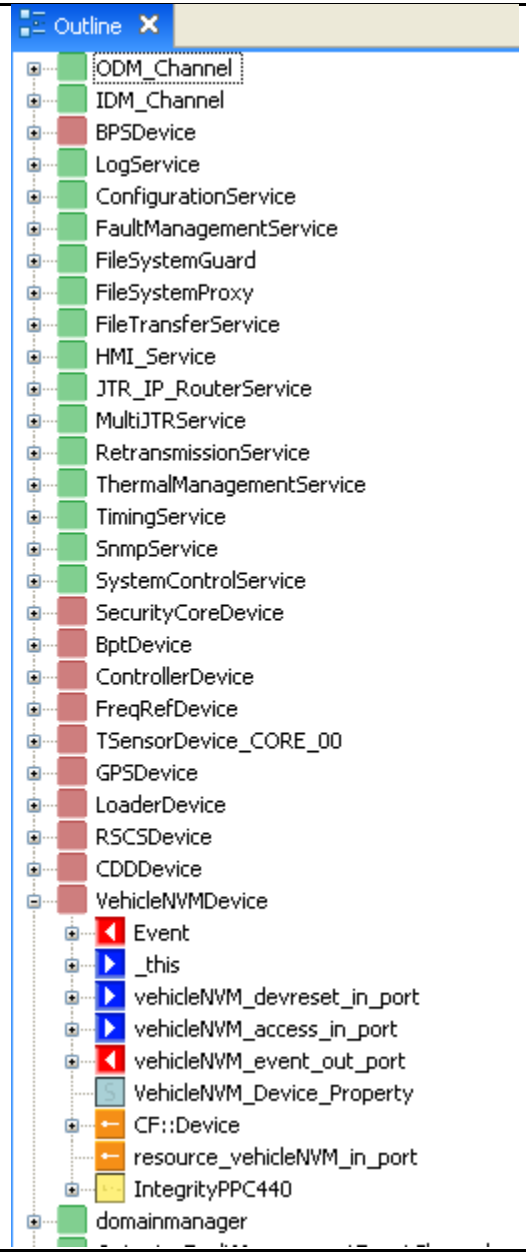
Manual Test Steps

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.

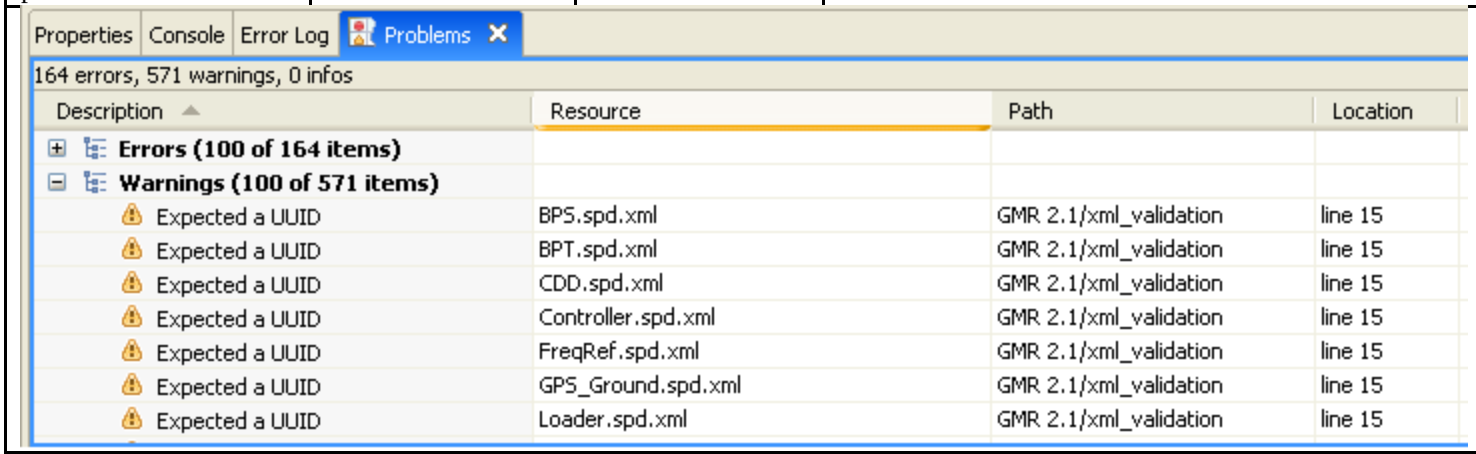
OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Set up a workspace for the OE and import the developer's files.				
1. Create a workspace directory in your My Documents.	A workspace directory is created.		A workspace is a location where Eclipse and Spectra keep all the files for your projects.	
2. Copy the OE files, including all the Domain Profile 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 Spectra.	Spectra starts.		You may get a window asking you to select a workspace. If so, select the workspace directory created in step 1 and skip steps 4 & 5.	
4. Select **File > Switch Workspace**				
5. Browse to the workspace created in step 1 and select it.				
6. Select **window > preferences**				
7. Select Spectra License Preferences and verify that the current license is listed. If it isn't, browse to it and select it.			The first time a workspace is built, it seems that Spectra does not know where the license file is.	
B. Build an SCA Platform Project.				

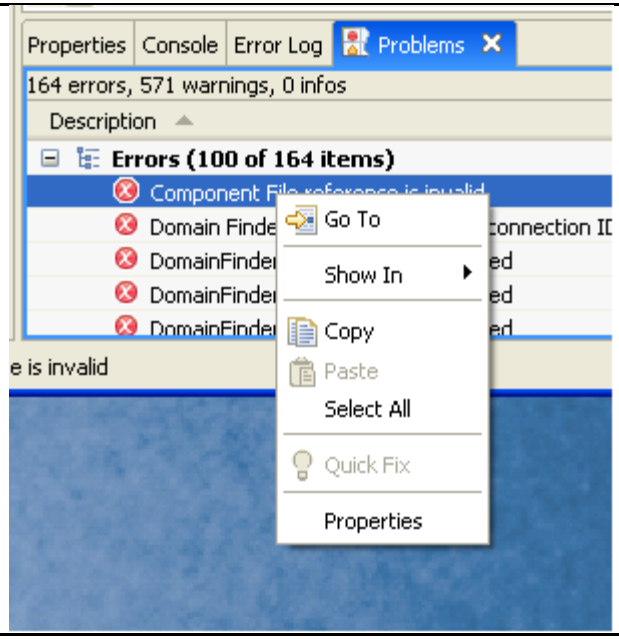
OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
8. Select **File > New > Project**.	A Project Wizard Window will open.			
9. Select **SCA Platform Project**.	The window will update to a New SCA Project Window.		Do **not** select SCA Project. An SCA project is an application, not an OE.	
10. Select **Create an SCA Platform Project from Domain Profile xml**, then **Next>**.	The window will change to a Name Project window.			
11. Name the project and select next.	The window will change to a Select DMD File window.			
12. Browse to the DMD file, select it, then select next	The window will change to a Select DCD and IDLs window.		There may be several DMD files. Determine which one to use with the aid of the developer's engineer.	
13. Browse to the DCD file and select it.	The absolute name of the DCD file will appear in the window.			

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
14. 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.			
15. Check the box for search all subfolders, then select finish.	The XML and IDL files will be parsed and the Platform project created.		If there is more than one DCD file that needs to be processed, select next instead of finish. This will give you the opportunity to enter another DCD file.	
C. Build the device, service and port lists.				
16. Select **open Assembly** and select **PlatformAssembly.ptf** 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.	

17. In the Outline window expand each device and service.				

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
18. Record the devices listed.	A list of devices will be recorded.		Devices are a pale violet.	
19. 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.	
20. Record the services listed.	A list of services will be presented.		Services are green.	
D. Record all errors and warnings.				
21. Select the **Error Log** tab then highlight an error.	An error in the error list will be highlighted.		The error list may need to be expanded to do this (click on the "+" in the tree.	
22. Right click and select **export log**	An Export data window will pop up.			
23. Name the log and select the directory in which you want to put it. Record the name. Then select **save**.			Note that most of these errors will not be of interest to JTEL testing. They are being saved for documentation purposes and reporting problems to PrismTech.	
24. Select the problems tab and then highlight a problem.			These errors and warnings indicate problems with the Domain Profile files. They will be analyzed for SCA errors.	



OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
25. Expand both the error and problem lists, then highlight a problem.	A Problem will be highlighted.			
26. Right click and choose **select all**, then right click again and select **copy**.	All problems will be highlighted.			
27. Open notepad and press **ctrl-V** (->edit->paste).	The problems list will be pasted into notepad.			
28. Save and close the notepad file. Record the name.	The file will be saved.		This file can be imported into a spreadsheet to help with error analysis.	
29. Review the problems log and determine if the problems are SCA related.	**Pass**: None of the problems are SCA related. (OE0613) **Fail**: One or more problems are SCA related. (OE0613)		Double clicking on a problem in the Spectra log will open the file and take you to the line that has the problem.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
For all criteria steps below, success is based on not finding the issue mentioned in the problems log.				
30. Within the problems log verify that there is no mention that the SCD file has failed to define the CORBA interfaces supported and used by a specific component.	**Pass**: The log makes no mention of this. (OE0613-C146) **Fail**: The log makes mention of this. (OE0613-C146)		Failure of OE0613-C146 results in failure of the OE0613 requirement. The Software Component Descriptor (SCD) defines the CORBA interfaces supported and used by a specific component.	
31. Within the problems log verify that there is no mention that the SPD file has failed to reference component level PRF file and SCD file.	**Pass**: The log makes no mention of this. (OE0613-C147) **Fail**: The log makes mention of this. (OE0613-C147)		Failure of OE0613-C147 results in failure of the OE0613 requirement. Within the specification of a software package descriptor several other files are referenced including a component level property file and a software component descriptor file. Within any given implementation there may be additional property files.	
32. During the SPD XML validation, search problems log file for the string, "Invalid softpkg id", SPD error #17. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C148) **Fail**: An instance of the string was found. (OE0613-C148)		Failure of OE0613-C148 results in failure of the OE0613 requirement. 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.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
33. During the SCD XML validation, search problems log file for the string, "Referenced file ??? does not exist on the file system", SCD error #3. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C149) **Fail**: An instance of the string was found. (OE0613-C149)		Failure of OE0613-C149 results in failure of the OE0613 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.	
34. 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.", SPD error #28. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C150) **Fail**: An instance of the string was found. (OE0613-C150)		Failure of OE0613-C150 results in failure of the OE0613 requirement. 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.	
35. 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: ???", SPD error #29. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C150) **Fail**: An instance of the string was found. (OE0613-C150)		Failure of OE0613-C150 results in failure of the OE0613 requirement. "???" represents the name of the related SCD.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
36. Within the problems log verify that there is no mention that duplicate properties had a type mismatch.	**Pass**: The log makes no mention of this. (OE0613-C151) **Fail**: The log makes mention of this. (OE0613-C151)		Failure of OE0613-C151 results in failure of the OE0613 requirement. Duplicated properties must be the same property type, only the value can be over-ridden.	
37. 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.	**Pass**: The log makes no mention of this. (OE0613-C152) **Fail**: The log makes mention of this. (OE0613-C152)		Failure of OE0613-C152 results in failure of the OE0613 requirement. 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.	
38. During the SPD XML validation, search problems log file for the string, "Referenced Property file is not a valid Properties XML file.", SPD error #9. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C153) **Fail**: An instance of the string was found. (OE0613-C153)		Failure of OE0613-C153 results in failure of the OE0613 requirement. The propertyfile element is used to indicate the local filename of the Property Descriptor file associated with the Software Package.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
39. During the SPD XML validation, search problems log file for the string, "Reference file (e.g., **SCD, Property, Implementation SPD, Implementation Code, Implementation Property file**) does not exist on FileSystem.", SPD error #11. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C154) **Fail**: An instance of the string was found. (OE0613-C154)		Failure of OE0613-C154 results in failure of the OE0613 requirement. 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.	
40. Within the problems log verify that there is no mention that the SCD file is required.	**Pass**: The log makes no mention of this. (OE0613-C155) **Fail**: The log makes mention of this. (OE0613-C155)		Failure of OE0613-C155 results in failure of the OE0613 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).	
41. 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.	**Pass**: The log makes no mention of this. (OE0613-C156) **Fail**: The log makes mention of this. (OE0613-C156)		Failure of OE0613-C156 results in failure of the OE0613 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.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
42. During the SPD XML validation, search problems log file for the string, "Invalid Implementation ID.???", SPD error #21. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C157) **Fail**: An instance of the string was found. (OE0613-C157)		Failure of OE0613-C157 results in failure of the OE0613 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.	
43. During the SPD XML validation, search problems log file for the string, "Invalid code stacksize value specified for implementation with id:???", SPD error#13. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C158) **Fail**: An instance of the string was found. (OE0613-C158)		Failure of OE0613-C158 results in failure of the OE0613 requirement. The stack size and priority are options parameters used by the CF ExecutableDevice execute() operation. "???" represents the name of the related SPD.	
44. During the SPD XML validation, search problems log file for the string, "Invalid code priority value specified for implementation with id:???", SPD error#14. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C158) **Fail**: An instance of the string was found. (OE0613-C158)		Failure of OE0613-C158 results in failure of the OE0613 requirement. The stack size and priority are options parameters used by the CF ExecutableDevice execute() operation. "???" represents the name of the related SPD.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
45. Within the problems log verify that there is no mention that the stacksize and priority option parameters are not unsigned long values.	**Pass**: The log makes no mention of this. (OE0613-C159) **Fail**: The log makes mention of this. (OE0613-C159)		Failure of OE0613-C159 results in failure of the OE0613 requirement. Data types for the values of these options are unsigned long.	
46. During the SPD XML validation, search problems log file for the string, "Invalid code priority value specified for implementation with id:???", SPD error#12. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C160) **Fail**: An instance of the string was found. (OE0613-C160)		Failure of OE0613-C160 results in failure of the OE0613 requirement. 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." "???" represents the name of the related SPD.	
47. During the SPD XML validation, search problems log file for the string, "Dependency SDP Implementation is not Compatible with specified Implementation", SPD error #3. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C161) **Fail**: An instance of the string was found. (OE0613-C161)		Failure of OE0613-C161 results in failure of the OE0613 requirement. 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. **At the time of this writing, May 2009, this error is not supported by the Spectra v1.2.**	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
48. During the SPD XML validation, search problems log file for the string, "Referenced Softpkg is not a softpkg xml file.", SPD error #10. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C161) **Fail**: An instance of the string was found. (OE0613-C161)		Failure of OE0613-C161 results in failure of the OE0613 requirement. **At the time of this writing, May 2009, this error is footnoted that the error message is wrong.**	
49. During the SPD XML validation, search problems log file for the string, "The specified Dependency SDP Implementation is not Compatible to implementation", SPD error #27. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C161) **Fail**: An instance of the string was found. (OE0613-C161)		Failure of OE0613-C161 results in failure of the OE0613 requirement. **At the time of this writing, May 2009, this error is not supported by the Spectra v1.2.**	
50. During the SPD XML validation, search problems log file for the string, "The specified Dependency SDP Implementation is not found", SPD error #30. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C161) **Fail**: An instance of the string was found. (OE0613-C161)		Failure of OE0613-C161 results in failure of the OE0613 requirement. **At the time of this writing, May 2009, this error is not supported by the Spectra v1.2.**	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
51. 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.	**Pass**: The log makes no mention of this. (OE0613-C161) **Fail**: The log makes mention of this. (OE0613-C161)		Failure of OE0613-C161 results in failure of the OE0613 requirement.	
52. During the SPD XML validation, search problems log file for the string, "Uses device Propertyref refid is not a UUID.", SPD error #22. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C162) **Fail**: An instance of the string was found. (OE0613-C162)		Failure of OE0613-C162 results in failure of the OE0613 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.	
53. During the SPD XML validation, search problems log file for the string, "Uses device Propertyref must have a value", SPD error #23. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C162) **Fail**: An instance of the string was found. (OE0613-C162)		Failure of OE0613-C162 results in failure of the OE0613 requirement. **At the time of this writing, May 2009, this error is footnoted with the following:** 1. Multiple warnings for implementation uses device element property value missing error 2. SPD Uses Device element property value missing error not detected.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
54. During the SPD XML validation, search problems log file for the string, "Implementation dependency Propertyref value is empty.", SPD error #24. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C162) **Fail**: An instance of the string was found. (OE0613-C162)		Failure of OE0613-C162 results in failure of the OE0613 requirement.	
55. During the SPD XML validation, search problems log file for the string, "Implementation dependency Propertyref refid is not a UUID", SPD error #25. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C162) **Fail**: An instance of the string was found. (OE0613-C162)		Failure of OE0613-C162 results in failure of the OE0613 requirement.	
56. 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.	**Pass**: The log makes no mention of this. (OE0613-C163) **Fail**: The log makes mention of this. (OE0613-C163)		Failure of OE0613-C163 results in failure of the OE0613 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.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
57. During the PRF XML validation, search problems log file for the string, "Invalid ID for Property:???. Expected a UUID value", PRF error #20. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C164) **Fail**: An instance of the string was found. (OE0613-C164)		Failure of OE0613-C164 results in failure of the OE0613 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.	
58. 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.	**Pass**: The log makes no mention of this. (OE0613-C165) **Fail**: The log makes mention of this. (OE0613-C165)		Failure of OE0613-C165 results in failure of the OE0613 requirement. Only simple elements can be used as execparam types.	
59. 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.	**Pass**: The log makes no mention of this. (OE0613-C166) **Fail**: The log makes mention of this. (OE0613-C166)		Failure of OE0613-C166 results in failure of the OE0613 requirement. At a minimum, the component interface has to be a CF Resource, CF ResourceFactory, or CF Device interface.	
60. During the DMD XML validation, search problems log file for the string, "Referenced SPD File is not a SPD File Type", DMD error #5. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C178) **Fail**: An instance of the string was found. (OE0613-C178)		Failure of OE0613-C178 results in failure of the OE0613 requirement. The implementation of the domain manager is itself described by the Domain Manager Configuration Descriptor (DMD), which provides the location of the (SPD) file for the specific DomainManager implementation to be loaded.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
61. During the DMD XML validation, search problems log file for the string, "Referenced SPD file _ does not exist in file system", DMD error#6. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C178) **Fail**: An instance of the string was found. (OE0613-C178)		Failure of OE0613-C178 results in failure of the OE0613 requirement.	
62. During the DPD XML validation, search problems log file for the string, "Device Package ID not a UUID.", DPD error #9. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C179) **Fail**: An instance of the string was found. (OE0613-C179)		Failure of OE0613-C179 results in failure of the OE0613 requirement. The devicepkg id attribute uniquely identifies the package and is a DCE UUID, as defined in paragraph D.2.1.	
63. During the DPD XML validation, search problems log file for the string, "Hwdeviceregistration ID not a UUID.", DPD error #10. A search for a partial string may be sufficient to exclude this message.	**Pass**: No instance of the string was found. (OE0613-C180) **Fail**: An instance of the string was found. (OE0613-C180)		Failure of OE0613-C180 results in failure of the OE0613 requirement. The hwdeviceregistration id attribute uniquely identifies the device and is a DCE UUID, as defined in paragraph D.2.1.	
64. Within the problems log verify that there is no mention during Profile Description XML validation that the profile element does not specify an absolute pathname relative to the mounted CF FileSystem.	**Pass**: The log makes no mention of this. (OE0613-C181) **Fail**: The log makes mention of this. (OE0613-C181)		Failure of OE0613-C181 results in failure of the OE0613 requirement. The profile element is used to specify an absolute file pathname relative to a mounted CF FileSystem.	

OE_TC_119				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
E. Verify that the appropriate Domain Profile files are present. (OE0613)				
65. Verify that there are no errors related to missing XML files.	**Pass**: No XML files are missing. (OE0613) **Fail**: XML files are missing. (OE0613)		Spectra will complain if executable files are missing. This is not an SCA violation.	
End Of Test				

Test Recording Log – OE_TC_119						
Step 18 (Devices)	Step 19 (Ports)	Step 20 (Services)	Step 23 (Error Log File)	Step 28 (Problem Log File)	Step 29 (Errors)	Step 65 (XML files missing)

Test Recording Log – OE0119						
Step 30 (OE0613-C146)	Step 31 (OE0613-C147)	Step 32 (OE0613-C148)	Step 33 (OE0613-C149)	Step 34 (OE0613-C150)	Step 35 (OE0613-C150)	Step 36 (OE0613-C151)
Step 37 (OE0613-C152)	Step 38 (OE0613-C153)	Step 39 (OE0613-C154)	Step 40 (OE0613-C155)	Step 41 (OE0613-C156)	Step 42 (OE0613-C157)	Step 43 (OE0613-C158)
Step 44 (OE0613-C158)	Step 45 (OE0613-C159)	Step 46 (OE0613-C160)	Step 47 (OE0613-C161)	Step 48 (OE0613-C161)	Step 49 (OE0613-C161)	Step 50 (OE0613-C161)
Step 51 (OE0613-C161)	Step 52 (OE0613-C162)	Step 53 (OE0613-C162)	Step 54 (OE0613-C162)	Step 55 (OE0613-C162)	Step 56 (OE0613-C163)	Step 57 (OE0613-C164)
Step 58 (OE0613-C165)	Step 59 (OE0613-C166)	Step 60 (OE0613-C178)	Step 61 (OE0613-C178)	Step 62 (OE0613-C179)	Step 63 (OE0613-C180)	Step 64 (OE0613-C181)

Test Summary OE_TC_119

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

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

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

OE0613_____

OE0613-C146	_____	OE0613-C161	_____
OE0613-C147	_____	OE0613-C162	_____
OE0613-C148	_____	OE0613-C163	_____
OE0613-C149	_____	OE0613-C164	_____
OE0613-C150	_____	OE0613-C165	_____
OE0613-C151	_____	OE0613-C166	_____
OE0613-C152	_____	OE0613-C178	_____
OE0613-C153	_____	OE0613-C179	_____
OE0613-C154	_____	OE0613-C180	_____
OE0613-C155	_____	OE0613-C181	_____
OE0613-C156	_____		
OE0613-C157	_____		
OE0613-C158	_____		

Failed Items (Section/Step Number):

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.18. OE_TC_131 - AEP mandatory functions

Test Case Number: OE_TC_131
OS::AEP

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0657	The functions in Table B-3 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0658	The functions listed in Table B-4 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0659	The functions listed in Table B-5 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0660	The functions listed in Table B-6 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0663	The functions listed in Table B-8 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0664	The functions listed in Table B-9 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0665	The functions listed in Table B-10 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0667	The functions listed in Table B-11 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0668	The functions listed in Table B-12 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0669	The functions listed in Table B-13 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0670	The functions listed in Table B-14 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0671	The function listed in Table B-15 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0672	The function listed in Table B-16 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0673	The functions listed in Table B-17 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0762	The functions listed in Table B-7 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0763	The functions listed in Table B-18 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0764	The functions listed in Table B-19 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0765	The functions listed in Table B-20 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0767	The functions listed in Table B-21 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0768	The functions listed in Table B-22 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table B-1.
OE0769	The options, limits, and any other constraints on POSIX.1 shall be provided as described in Table B-2
OE0770	The function listed in Table B-23 shall behave as described in the referenced clause.

References

Document Name	Version/Date	Location (Pages, Section)
SCA Appendix B: AEP	Version 2.2.2 15 May 2006	Pages B-2 through B-18, Section B.4
1003.13TM IEEE Standard for Information Technology— Standardized Application Environment Profile (AEP)—POSIX® Realtime and Embedded Application Support	10 September 2004	Pages 111 thru 121, Annex B
The Open Group Base Specifications Issue 6, IEEE Std 1003.1	2004 Edition	http://www.opengroup.org/onlinepubs/000095399/

Test Objective

This test case verifies OE0657, OE0658, OE0659, OE0660, OE0663, OE0664, OE0665, OE0667, OE0668, OE0669, OE0670, OE0671, OE0672, OE0673, OE0762, OE0763, OE0764, OE0765, OE0767, OE0768, OE0769, and OE0770. The objective of this test is to verify that the OE supplies the functions designated as mandatory in Appendix B of the SCA and to verify that the POSIX.1 options that are mandatory are provided.

Note that this test is for the original AEP requirements. Use OE_TC_172 if the AEP Amended requirements are to be verified.

Places to Verify

OS/C/C++ header files, including aio.h, ctype.h, dirent.h, fcntl.h, locale.h, math.h, mqueue.h, pthread.h, sched.h, semaphore.h, setjmp.h, signal.h, stdio.h, stdlib.h, string.h, sys/mman.h, sys/stat.h, sys/types.h, time.h, unistd.h, and utime.h.

IDL References

None.

Preconditions

- The OE header source code files are available.

Test Description

A. Identify the source code files that define POSIX interfaces. (OE0657, OE0658, OE0659, OE0660, OE0663, OE0664, OE0665, OE0667, OE0668, OE0669, OE0670, OE0671, OE0672, OE0673, OE0762, OE0763, OE0764, OE0765, OE0767, OE0768, OE0769, OE0770)
   1. **N/A:** There are no source code files. The interface was not defined.
   2. **Pass:** Source code files are identified.
   3. **Pass:** The developer is using a third party's certified POSIX interface.

For the following steps use the source code found above or the third party's detailed implementation.

B. Verify that the options listed as mandatory in table B.2 of Appendix B are provided**.** (OE0769)
   1. **Pass:** The mandatory options are provided.
   2. **Fail:** The mandatory options are not provided.

C. Verify that the mandatory functions are provided. (OE0657, OE0658, OE0659, OE0660, OE0663, OE0664, OE0665, OE0667, OE0668, OE0669, OE0670, OE0671, OE0672, OE0673, OE0762, OE0763, OE0764, OE0765, OE0767, OE0768, OE0770)
   1. **Pass:** The mandatory functions are provided.
   2. **Fail:** The mandatory functions are not provided.

Manual Test Steps

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

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that define POSIX interfaces. (OE0657, OE0658, OE0659, OE0660, OE0663, OE0664, OE0665, OE0667, OE0668, OE0669, OE0670, OE0671, OE0672, OE0673, OE0762, OE0763, OE0764, OE0765, OE0767, OE0768, OE0769, OE0770)				
1. Perform a search for POSIX interfaces in the OE source code provided by the developer. Record the source file name(s).	**N/A**: The OE does not define POSIX interfaces. **Pass**: Source code files are found. **Pass**: The developer is using a third party's certified POSIX interface. (OE0657, OE0658, OE0659, OE0660, OE0663, OE0664, OE0665, OE0667, OE0668, OE0669, OE0670, OE0671, OE0672, OE0673, OE0762, OE0763, OE0764, OE0765, OE0767, OE0768, OE0769, OE0770)			
For the following steps use the source code found above or the third party's detailed implementation.				
B. Verify that the options listed as mandatory in table B.2 of Appendix B are provided. (OE0769)				
2. Verify that the functions of the following options are provided in the OE header files. The rows below from here to Step 2 are the mandatory options. The comments column discussed what must be present for that option to be supported.	**Pass**: All options provided. (OE0769) **Fail**: One or more options not provided. (OE0769)		**Step 2** is in several sections to separate the function signatures for easier reading. More information for each function can be found on this website: http://www.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap02.html#tag_02_01_06	
POSIX_ASYNCHRONOUS_IO			#include <aio.h>	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**

For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			int aio_cancel(int *fildes*, struct aiocb **aiocbp*);
			
			int aio_error(const struct aiocb **aiocbp*);
			
			int aio_fsync(int *op*, struct aiocb **aiocbp*
			
			int aio_read(struct aiocb **aiocbp*);
			
			ssize_t aio_return(struct aiocb **aiocbp*);
			
			int aio_suspend(const struct aiocb * const *list*[], int *nent*, const struct timespec ***timeout*);
			
			int aio_write(struct aiocb **aiocbp*);
			
			int lio_listio(int *mode*, struct aiocb *restrict const *list*[restrict], int *nent*, struct sigevent *restrict *sig*);	

POSIX_MEMLOCK_RANGE For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <sys/mman.h> int **mlock**(const void * *addr*, size_t *len*); int **munlock**(const void * *addr*, size_t *len*);	
POSIX_MEMLOCK For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <sys/mman.h> int **mlockall**(int *flags*); int **munlockall**(void);	
POSIX_MESSAGE_PASSING			#include <mqueue.h>	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			int **mq_close**(mqd_t *mqdes*); int **mq_getattr**(mqd_t *mqdes*, struct mq_attr **mqstat*); int **mq_notify**(mqd_t *mqdes*, const struct sigevent **notification*); mqd_t **mq_open**(const char **name*, int *oflag*, ...); ssize_t **mq_receive**(mqd_t *mqdes*, char **msg_ptr*, size_t *msg_len*, unsigned int **msg_prio*); int **mq_send**(mqd_t *mqdes*, const char **msg_ptr*, size_t *msg_len*, unsigned int *msg_prio*); int **mq_setattr**(mqd_t *mqdes*, const struct mq_attr **mqstat*, struct mq_attr **omqstat*); int **mq_unlink**(const char***name*);	
POSIX_REALTIME_SIGNALS For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <sys/types.h> #include <signal.h> int **sigqueue**(pid_t *pid*, int *signo*, const union sigval *value*); int **sigwaitinfo**(const sigset_t **set*, siginfo_t **info*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **sigtimedwait**(const sigset_t *_set_, siginfo_t *_info_ const struct timespec *_timeout_);	
POSIX_SEMAPHORES For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <semaphore.h> int **sem_close**(sem_t *_sem_); int **sem_destroy**(sem_t *_sem_); int **sem_getvalue**(sem_t *_sem_, int *_sval_); int **sem_init**(sem_t *_sem_, int _pshared_, unsigned int _value_); sem_t *__sem_open__(const char*_name_, int _oflag_, ...); int **sem_post**(sem_t *_sem_); int **sem_trywait**(sem_t *_sem_); int **sem_wait**(sem_t *_sem_); int **sem_unlink**(const char *_name_);	
POSIX_THREAD_ATTR_STACKADDR For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <pthread.h> int **pthread_attr_getstackaddr**(const pthread_attr_t *restrict _attr_, void **restrict _stackaddr_); int **pthread_attr_setstackaddr**(pthread_attr_t *_attr_, void *_stackaddr_);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
POSIX_THREAD_ATTR_STACKSIZE For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <pthread.h> int **pthread_attr_getstack**(const pthread_attr_t *restrict *attr*, void **restrict *stackaddr*, size_t *restrict *stacksize*); int **pthread_attr_setstack**(pthread_attr_t **attr*, void **stackaddr*, size_t *stacksize*);	
POSIX_THREAD_PRIO_INHERIT For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <pthread.h> int **pthread_mutexattr_getprotocol**(const pthread_mutexattr_t *restrict *attr*, int *restrict *protocol*); int **pthread_mutexattr_setprotocol**(pthread_mutexattr_t **attr*, int *protocol*);	
POSIX_THREAD_PRIO_PROTECT For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <pthread.h> int **pthread_mutex_getprioceiling**(const pthread_mutex_t *restrict *mutex*, int *restrict *prioceiling*); int **pthread_mutex_setprioceiling**(pthread_mutex_t *restrict *mutex*, int *prioceiling*, int *restrict *old_ceiling*); int **pthread_mutexattr_getprioceiling**(con	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			st pthread_mutexattr_t *restrict *attr*, int *restrict *prioceiling*); int **pthread_mutexattr_setprioceiling**(pthread_mutexattr_t *_attr_, int _prioceiling_); int **pthread_mutexattr_getprotocol**(const pthread_mutexattr_t *restrict *attr*, int *restrict *protocol*); int **pthread_mutexattr_setprotocol**(pthread_mutexattr_t *_attr_, int _protocol_);	
POSIX_THREAD_PRIORITY_SCHEDULING For the above option to be supported by the OE, the utilities, functions and facilities in the Comments column must be present.			#include <pthread.h> int **pthread_attr_getinheritsched**(const pthread_attr_t *restrict *attr*, int *restrict *inheritsched*); int **pthread_attr_setinheritsched**(pthread_attr_t *_attr_, int _inheritsched_); int **pthread_attr_getscope**(const pthread_attr_t *restrict *attr*, int *restrict *contentionscope*); int **pthread_attr_setscope**(pthread_attr_t *_attr_, int _contentionscope_);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **pthread_attr_setschedpolicy**(pthread_attr_t *_attr_, int _policy_); int **pthread_attr_getschedpolicy**(const pthread_attr_t*_attr_, int *_policy_); int **pthread_getschedparam**(pthread_t _thread_, int *_policy_, struct sched_param*_param_); int **pthread_setschedparam**(pthread_t _thread_, int _policy_, const struct sched_param*_param_); int **pthread_setschedprio**(pthread_t _thread_, int _prio_); #include <sched.h> int **sched_get_priority_max**(int _policy_); int **sched_get_priority_min**(int _policy_); int **sched_rr_get_interval**(pid_t _pid_, struct timespec*_interval_);	
POSIX_TIMERS For the above option to be supported by the OE, the utilities, functions and			#include <time.h>	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
facilities in the Comments column must be present.			int **clock_getres**(clockid_t *clock_id*, struct timespec *_res_); int **clock_gettime**(clockid_t *clock_id*, struct timespec *_tp_); int **clock_settime**(clockid_t *clock_id*, const struct timespec *_tp_); int **nanosleep**(const struct timespec *_rqtp_, struct timespec *_rmtp_); #include <signal.h> int timer_**create**(clockid_t *clockid*, struct sigevent *_evp_, timer_t *_timerid_); int timer_**delete**(timer_t *timerid*); int timer_**getoverrun**(timer_t *timerid*); int timer_**gettime**(timer_t *timerid*, struct itimerspec *_value_); int timer_**settime**(timer_t *timerid*, int *flags*, const struct itimerspec *restrict *value*, struct itimerspec *restrict *ovalue*);	
C. Verify that The mandatory functions are provided. (OE0657, OE0658, OE0659, OE0660, OE0663, OE0664, OE0665, OE0667, OE0668, OE0669, OE0670, OE0671, OE0672, OE0673, OE0762, OE0763, OE0764, OE0765, OE0767, OE0768, OE0770)				
3. Verify that the following functions from table B-3 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0657) **Fail**: The mandatory functions are not provided. (OE0657)		There are no mandatory functions listed in table B-3. All OE's pass this requirement.	
4. Verify that the following functions from table B-4 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0658)		There are no mandatory functions listed in table B-4. All OE's pass this requirement.	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
	Fail: The mandatory functions are not provided. (OE0658)			
5. Verify that the following functions from table B-5 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0659) **Fail**: The mandatory functions are not provided. (OE0659)		There are no mandatory functions listed in table B-5. All OE's pass this requirement.	
6. Verify that the following functions from table B-6 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0660) **Fail**: The mandatory functions are not provided. (OE0660)		**POSIX_SIGNALS** #include <stdlib.h> void **abort**(void); #include <unistd.h> int **pause**(void); #include <signal.h> int **kill**(pid_t, int); int **raise**(int *sig*); int **sigaction**(int *sig*, const struct sigaction *restrict *act*, struct sigaction *restrict *oact*); int **sigaddset**(sigset_t **set*, int *signo*); int **sigdelset**(sigset_t **set*, int *signo*); int **sigemptyset**(sigset_t **set*); int **sigfillset**(sigset_t **set*); int **sigismember**(const sigset_t **set*, int *signo*); void (***signal**(int *sig*, void (**func*)(int)))(int);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **sigpending**(sigset_t *_set_); int **sigprocmask**(int _how_, const sigset_t *restrict _set_, sigset_t *restrict _oset_); int pthread_**sigmask**(int _how_, const sigset_t *restrict _set_, sigset_t *restrict _oset_); int **sigsuspend**(const sigset_t *_sigmask_); int **sigwait**(const sigset_t *restrict _set_, int *restrict _sig_);	
7. Verify that the following functions from table B-7 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0762) **Fail**: The mandatory functions are not provided. (OE0762)		There are no mandatory functions listed in table B-7. All OE's pass this requirement.	
8. Verify that the following functions from table B-8 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0663) **Fail**: The mandatory functions are not provided. (OE0663)		There are no mandatory functions listed in table B-8. All OE's pass this requirement.	
9. Verify that the following functions from table B-9 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0664) **Fail**: The mandatory functions are not provided. (OE0664)		**POSIX_FILE_SYSTEM** #include <fcntl.h> int **creat**(const char *_path_, mode_t _mode_); #include <sys/stat.h> int **mkdir**(const char *_path_, mode_t _mode_); int **fstat**(int _fildes_, struct stat *_buf_);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**

			#include <unistd.h>
			int **access**(const char *_path_, int _amode_);
			
			int **chdir**(const char *_path_);
			
			long **fpathconf**(int _fildes_, int _name_);
			
			long **pathconf**(const char *_path_, int _name_);
			
			char *__**getcwd**__(char *_buf_, size_t _size_);
			
			int **link**(const char*_path1_, const char *_path2_);
			
			int **rmdir**(const char *_path_);
			
			int **unlink**(const char *_path_);
			
			#include <dirent.h>
			int **closedir**(DIR *_dirp_);
			
			DIR *__**opendir**__(const char *_dirname_);
			
			struct **dirent***readdir(DIR *_dirp_);
			
			int **readdir_r**(DIR *restrict _dirp_, struct dirent *restrict _entry_, struct dirent **restrict _result_);
			
			void **rewinddir**(DIR *_dirp_);
			
			#include <stdio.h>
			int **remove**(const char*_path_);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **rename**(const char *_old_, const char *_new_); FILE ***tmpfile**(void); char ***tmpnam**(char*_s_); #include <sys/stat.h> int **stat**(const char *restrict _path_, struct stat *restrict _buf_); #include <utime.h> int **utime**(const char *_path_, const struct utimbuf*_times_);	
10. Verify that the following functions from table B-10 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0665) **Fail**: The mandatory functions are not provided. (OE0665)		There are no mandatory functions listed in table B-10. All OE's pass this requirement.	
11. Verify that the following functions from table B-11 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0667) **Fail**: The mandatory functions are not provided. (OE0667)		**POSIX_FD_MGMT** #include <stdio.h> int **fseek**(FILE *_stream_, long _offset_, int _whence_); int **fseeko**(FILE *_stream_, off_t _offset_, int _whence_); void **rewind**(FILE *_stream_); long **ftell**(FILE *_stream_); off_t **ftello**(FILE *_stream_);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			#include <unistd.h> off_t **lseek**(int *fildes*, off_t *offset*, int *whence*);	

12. Verify that the following functions from table B-12 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0668)
	
	Fail: The mandatory functions are not provided. (OE0668)
			POSIX_DEVICE_IO
			#include <fcntl.h>
			
			int **open**(const char *_path_, int _oflag_, ...);
			
			#include <stdio.h>
			
			int **getc**(FILE *_stream_);
			
			int **getchar**(void);
			
			char *__gets__(char *_s_);
			
			int **fgetc**(FILE *_stream_);
			
			char *__fgets__(char *restrict _s_, int _n_, FILE *restrict _stream_);
			
			int **fileno**(FILE *_stream_);
			
			FILE *__fopen__(const char *restrict _filename_, const char *restrict _mode_);
			
			int **fprintf**(FILE *restrict _stream_, const char *restrict _format_, ...);
			
			int **fputc**(int _c_, FILE *_stream_);
			
			int **fputs** (const char *restrict _s_, FILE *restrict _stream_);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			size_t **fread**(void *restrict *ptr*, size_t *size*, size_t *nitems*, FILE *restrict *stream*); FILE ***freopen**(const char*restrict int **fscanf**(FILE *restrict *stream*, const char *restrict *format*, ...); size_t **fwrite**(const void *restrict *ptr*, size_t *size*, size_t *nitems*, FILE *restrict *stream*); void **perror**(const char *s); int **putc**(int *c*, FILE **stream*); int **putchar**(int *c*); int **puts**(const char**s*); int **printf**(const char *restrict *format*, ...); int **snprintf**(char *restrict *s*, size_t *n*, const char *restrict *format*, ...); int **sprintf**(char *restrict *s*, const char *restrict *format*, ...); int **scanf**(const char *restrict *format*, ...); int **sscanf**(const char *restrict *s*, const char *restrict *format*, ...); #include <unistd.h> int **close**(int *fildes*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			void **clearerr**(FILE *_stream_); int **fclose**(FILE *_stream_); FILE *__fdopen__(int _fildes_, const char *_mode_); int **feof**(FILE *_stream_); int **ferror**(FILE *_stream_); int **fflush**(FILE *_stream_); ssize_t **read**(int _fildes_, void *_buf_, size_t _nbyte_); void **setbuf**(FILE *restrict _stream_, char *restrict _buf_); #include <stdio.h> int **setvbuf**(FILE *restrict _stream_, char *restrict _buf_, int _type_, size_t _size_); int **ungetc**(int _c_, FILE *_stream_); ssize_t **write**(int _fildes_, const void *_buf_, size_t _nbyte_);	
13. Verify that the following functions from table B-13 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0669) **Fail**: The mandatory functions are not provided. (OE0669)		There are no mandatory functions listed in table B-13. All OE's pass this requirement.	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
14. Verify that the following functions from table B-14 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0670) **Fail**: The mandatory functions are not provided. (OE0670)		There are no mandatory functions listed in table B-14. All OE's pass this requirement.	
15. Verify that the following functions from table B-15 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0671) **Fail**: The mandatory functions are not provided. (OE0671)		There are no mandatory functions listed in table B-15. All OE's pass this requirement.	
16. Verify that the following functions from table B-16 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0672) **Fail**: The mandatory functions are not provided. (OE0672)		There are no mandatory functions listed in table B-16. All OE's pass this requirement.	
17. Verify that the following functions from table B-17 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0673) **Fail**: The mandatory functions are not provided. (OE0673)		**POSIX_C_LANG_SUPPORT** #include <stdlib.h> int **abs**(int *i*); double **atof**(const char **str*); int **atoi**(const char **str*); long **atol**(const char***str*); long long **atoll**(const char***nptr*); void ****bsearch**(const void ***key*, const void ***base*, size_t *nel*, size_t *width*, int (***compar*)(const void *, const void *)); void ****calloc**(size_t *nelem*, size_t *elsize*); void **free**(void ***ptr*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **fscanf**(FILE *restrict *stream*, const char *restrict *format*, ...); void ***malloc**(size_t *size*); void **qsort**(void **base*, size_t *nel*, size_t *width*, int (**compar*)(const void *, const void *)); int **rand**(void); int **rand_r**(unsigned **seed*); *timer*, struct tm *restrict *result*); time_t **mktime**(struct tm **timeptr*); struct tm ***localtime**(const time_t *restrict *timer*, struct tm *restrict *result*); size_t **strftime**(char *restrict *s*, size_t *maxsize*, const char *restrict *format*, const struct tm *restrict *timeptr*); time_t **time**(time_t **tloc*); #include <ctype.h> int **isalnum**(int *c*); int **isalpha**(int *c*); int **iscntrl**(int *c*); int **isdigit**(int *c*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **isgraph**(int *c*); int **islower**(int *c*); int **isprint**(int *c*); int **ispunct**(int *c*); int **isspace**(int *c*); int **isupper**(int *c*); int **isxdigit**(int *c*); void ***realloc**(void **ptr*, size_t *size*); int **scanf**(const char *restrict *format*, ...); int **sscanf**(const char *restrict *s*, const char *restrict *format*, ...); #include <time.h> char ***asctime**(const struct tm **timeptr*); char ***ctime**(const time_t **clock*); char ***ctime_r**(const time_t **clock*,char **buf*); struct tm ***gmtime**(const time_t **timer*); struct tm ***gmtime_r**(const time_t *restrict int **tolower**(int *c*); int **toupper**(int *c*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**

			#include <locale.h>
			
			char *__setlocale__(int *category*, const char **locale*);
			
			#include <stdio.h>
			
			int __fprintf__(FILE *restrict *stream*, const char *restrict *format*, ...);
			
			int __printf__(const char *restrict *format*, ...);
			
			int __snprintf__(char *restrict *s*, size_t *n*, const char *restrict *format*, ...);
			
			int __sprintf__(char *restrict *s*, const char *restrict *format*, ...);
			
			#include <string.h>
			
			char *__strcat__(char *restrict *s1*, const char *restrict *s2*);
			
			char *__strchr__(const char *s, int *c*);
			
			int __strcmp__(const char **s1*, const char **s2*);
			
			char *__strcpy__(char *restrict *s1*, const char *restrict *s2*);
			
			size_t __strcspn__(const char **s1*, const char **s2*);
			
			int __strcmp__(const char **s1*, const char **s2*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**

			size_t **strlen**(const char *s);
			
			char *__**strncat**__(char *restrict *s1*, const char *restrict *s2*, size_t *n*);
			
			int **strncmp**(const char *s1, const char *s2, size_t *n*);
			
			char *__**strncpy**__(char *restrict *s1*, const char *restrict *s2*, size_t *n*);
			
			char *__**strpbrk**__(const char *s1, const char *s2);
			
			char *__**strrchr**__(const char *s, int *c*);
			
			size_t **strspn**(const char *s1, const char *s2);
			
			char *__**strstr**__(const char *s1, const char *s2);
			
			char *__**strtok**__(char *restrict *s1*, const char *restrict *s2*);
			
			char ***strtok_r**(char *restrict *s*, const char *restrict *sep*, char **restrict *lasts*);	

18. Verify that the following functions from table B-18 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0763)
	
	Fail: The mandatory functions are not provided. (OE0763)
			POSIX_C_LANG_MATH
			#include <math.h>
			
			double **acos**(double *x*);
			float acosf(float *x*);
			long double acosl(long double *x*);
			
			double **asin**(double *x*);
			float asinf(float *x*);
			long double asinl(long double *x*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**

			double **atan**(double *x*);
			float atanf(float *x*);
			long double atanl(long double *x*);
			
			double **atan2**(double *y*, double *x*);
			float atan2f(float *y*, float *x*);
			long double atan2l(long double *y*, long double *x*);
			
			double **ceil**(double *x*);
			float ceilf(float *x*);
			long double ceill(long double *x*);
			
			double **cos**(double *x*);
			float cosf(float *x*);
			long double cosl(long double *x*);
			
			double **cosh**(double *x*)
			float coshf(float *x*);
			long double coshl(long double *x*);
			
			double **exp**(double *x*);
			float expf(float *x*);
			long double expl(long double *x*);
			
			double **fabs**(double *x*);
			float fabsf(float *x*);
			long double fabsl(long double *x*);
			
			double **floor**(double *x*);
			float floorf(float *x*);
			long double floorl(long double *x*);
			
			double **fmod**(double *x*, double *y*);
			float fmodf(float *x*, float *y*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**

			long double fmodl(long double *x*, long double *y*);
			
			double **frexp**(double *num*, int **exp*);
			float frexpf(float *num*, int **exp*)
			long double frexpl(long double *num*, int **exp*);
			
			double **ldexp**(double *x*, int *exp*);
			float ldexpf(float *x*, int *exp*);
			long double ldexpl(long double *x*, int *exp*);
			
			double **log**(double *x*);
			float logf(float *x*);
			long double logl(long double *x*);
			
			double **log10**(double *x*);
			float log10f(float *x*);
			long double log10l(long double *x*);
			
			double **modf**(double *x*, double **iptr*);
			float modff(float *value*, float **iptr*);
			long double modfl(long double *value*, longdouble **iptr*);
			
			double **pow**(double *x*, double *y*);
			float powf(float *x*, float *y*);
			long double powl(long double *x*, long double *y*);
			
			double **sin**(double *x*);
			float sinf(float *x*);
			long double sinl(long double *x*);
			
			double **sinh**(double *x*);
			float sinhf(float *x*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**

			long double sinhl(long double *x*);
			
			double **sqrt**(double *x*);
			float sqrtf(float *x*);
			long double sqrtl(long double *x*);
			
			double **tan**(double *x*);
			float tanf(float *x*);
			long double tanl(long double *x*);
			
			double **tanh**(double *x*);
			float tanhf(float *x*);
			long double tanhl(long double *x*);	

19. Verify that the following functions from table B-19 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0764) **Fail**: The mandatory functions are not provided. (OE0764)		**POSIX_C_LANG_JUMP** #include <setjmp.h> void **longjmp**(jmp_buf *env*, int *val*); int **setjmp**(jmp_buf *env*);	
20. Verify that the following functions from table B-20 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0765) **Fail**: The mandatory functions are not provided. (OE0765)		**POSIX_SEMAPHORES** #include <semaphore.h> int **sem_close**(sem_t **sem*); int **sem_destroy**(sem_t **sem*); int **sem_getvalue**(sem_t **sem*, int **sval*); int **sem_init**(sem_t **sem*, int *pshared*, unsigned int *value*); sem_t ***sem_open**(const char **name*, int *oflag*, ...); int **sem_post**(sem_t **sem*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **sem_trywait**(sem_t *_sem_); int **sem_wait**(sem_t *_sem_); int **sem_unlink**(const char *_name_);	
21. Verify that the following functions from table B-21 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0767) **Fail**: The mandatory functions are not provided. (OE0767)		**POSIX_TIMERS** #include <time.h> int **clock_getres**(clockid_t _clock_id_, struct timespec *_res_); int **clock_gettime**(clockid_t _clock_id_, struct timespec *_tp_); int **clock_settime**(clockid_t _clock_id_, const struct timespec *_tp_); int **nanosleep**(const struct timespec *_rqtp_, struct timespec *_rmtp_); #include <signal.h> int timer_**create**(clockid_t _clockid_, struct sigevent *_evp_, timer_t *_timerid_); int timer_**delete**(timer_t _timerid_); int timer_**getoverrun**(timer_t _timerid_); int timer_**gettime**(timer_t _timerid_, struct itimerspec *_value_); int timer_**settime**(timer_t _timerid_, int _flags_, const struct itimerspec *restrict _value_, struct itimerspec *restrict _ovalue_);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
22. Verify that the following functions from table B-22 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0768) **Fail**: The mandatory functions are not provided. (OE0768)		**POSIX_THREADS_BASE** #include <pthread.h> int **pthread_attr_getschedparam**(const pthread_attr_t *restrict *attr*, struct sched_param *restrict *param*); int **pthread_attr_setschedparam**(pthred_attr_t *restrict *attr*, const struct sched_param *restrict *param*); int **pthread_cancel**(pthread_t *thread*); void **pthread_cleanup_pop**(int *execute*); void **pthread_cleanup_push**(void (**routine*)(void*), void **arg*); int **pthread_cond_destroy**(pthread_cond_t **cond*); int **pthread_cond_init**(pthread_cond_t *restrict *cond*, const pthread_condattr_t *restrict *attr*); int pthread_cond_t *cond* = PTHREAD_COND_INITIALIZER; int **pthread_cond_broadcast**(pthread_cond_t **cond*); int **pthread_cond_signal**(pthread_cond_t **cond*); int **pthread_cond_timedwait**(pthread_con	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			d_t *restrict *cond*, pthread_mutex_t *restrict *mutex*, const struct timespec *restrict *abstime*); int **pthread_cond_wait**(pthread_cond_t *restrict *cond*, pthread_mutex_t *restrict *mutex*); int **pthread_condattr_destroy**(pthread_condattr_t **attr*); int **pthread_condattr_init**(pthread_condattr_t **attr*); int **pthread_condattr_getclock**(const pthread_condattr_t *restrict *attr*, clockid_t *restrict *clock_id*); int **pthread_condattr_setclock**(pthread_condattr_t **attr*, clockid_t *clock_id*); int **pthread_condattr_getpshared**(const pthread_condattr_t *restrict *attr*, int *restrict *pshared*); int **pthread_condattr_setpshared**(pthread_condattr_t **attr*, int *pshared*); int **pthread_create**(pthread_t *restrict *thread*, const pthread_attr_t *restrict *attr*, void *(**start_routine*)(void*), void *restrict *arg*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **pthread_detach**(pthread_t *thread*); int **pthread_equal**(pthread_t *t1*, pthread_t *t2*); void **pthread_exit**(void **value_ptr*); int **pthread_getschedparam**(pthread_t *thread*, int *restrict policy*, struct sched_param *restrict *param*); int **pthread_setschedparam**(pthread_t *thread*, int *policy*, const struct sched_param ***param*); void ****pthread_getspecific**(pthread_key_t *key*); int **pthread_setspecific**(pthread_key_t *key*, const void ***value*); int **pthread_join**(pthread_t *thread*, void ****value_ptr*); int **pthread_key_create**(pthread_key_t ***key*, void (***destructor*)(void*)); int **pthread_key**_delete(pthread_key_t *key*); int **pthread_kill**(pthread_t *thread*, int *sig*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **pthread_mutexattr_destroy**(pthread_mutexattr_t *_attr_); int **pthread_mutexattr_init**(pthread_mutexattr_t *_attr_); int **pthread_mutexattr_getprioceiling**(const pthread_mutexattr_t *restrict _attr_, int *restrict _prioceiling_); int **pthread_mutexattr_setprioceiling**(pthread_mutexattr_t *_attr_, int _prioceiling_); int **pthread_mutexattr_getprotocol**(const pthread_mutexattr_t *restrict _attr_, int *restrict _protocol_); int **pthread_mutexattr_setprotocol**(pthread_mutexattr_t *_attr_, int _protocol_); int **pthread_mutexattr_getpshared**(const pthread_mutexattr_t *restrict _attr_, int *restrict _pshared_); int **pthread_mutexattr_setpshared**(pthread_mutexattr_t *_attr_, int _pshared_); int **pthread_mutexattr_gettype**(const pthread_mutexattr_t *restrict _attr_, int *restrict _type_);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **pthread_mutexattr_settype**(pthread_mutexattr_t **attr*, int *type*); int **pthread_mutex_destroy**(pthread_mutex_t **mutex*); int **pthread_mutex_init**(pthread_mutex_t *restrict *mutex*, const pthread_mutexattr_t *restrict *attr*); pthread_mutex_t *mutex* = PTHREAD_MUTEX_INITIALIZER; int **pthread_mutex_getprioceiling**(const pthread_mutex_t *restrict *mutex*, int *restrict *prioceiling*); int **pthread_mutex_setprioceiling**(pthread_mutex_t *restrict *mutex*, int *prioceiling*, int *restrict *old_ceiling*); int **pthread_mutex_lock**(pthread_mutex_t **mutex*); int **pthread_mutex_trylock**(pthread_mutex_t **mutex*); int **pthread_mutex_unlock**(pthread_mutex_t **mutex*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **pthread_mutex_timedlock**(pthread_mutex_t *restrict *mutex* const struct timespec *restrict *abs_timeout*); int **pthread_once**(pthread_once_t **once_control*, void (**init_routine*)(void)); pthread_once_t *once_control* = PTHREAD_ONCE_INIT; pthread_t **pthread_self**(void); int **pthread_setcancelstate**(int *state*, int **oldstate*); int **pthread_setcanceltype**(int *type*, int **oldtype*); void **pthread_testcancel**(void); int **pthread_getschedparam**(pthread_t *thread*, int *restrict *policy*, struct sched_param *restrict *param*); int **pthread_setschedparam**(pthread_t *thread*, int *policy*, const struct sched_param **param*); void *__**pthread_getspecific**(pthread_key_t *key*); int **pthread_setspecific**(pthread_key_t *key*, const void **value*);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			int **pthread_sigmask**(int *how*, const sigset_t *restrict *set*, sigset_t *restrict *oset*); int **sigprocmask**(int *how*, const sigset_t *restrict *set*, sigset_t *restrict *oset*);	

23. Verify that the following functions from table B-23 are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0770)
	
	Fail: The mandatory functions are not provided. (OE0770)
			If one or more functions listed below are not provided, then OE0770 fails this step.
			
			POSIX_THREAD_SAFE_FUNCTIONS
			#include <time.h>
			
			char *__asctime__(const struct tm *__timeptr__);
			
			char *__ctime__(const time_t *__clock__);
			
			struct tm *__gmtime__(const time_t *__timer__);
			
			struct tm *__localtime__(const time_t *__timer__);
			struct tm *__localtime_r__(const time_t *restrict __timer__, struct tm *restrict __result__);
			
			#include <stdlib.h>
			
			int __rand__(void);
			int __rand_r__(unsigned *__seed__);
			
			#include <dirent.h>
			struct __dirent__ *readdir(DIR *__dirp__);
			
			int __readdir_r__(DIR *restrict __dirp__, struct dirent *restrict __entry__, struct dirent **restrict __result__);	

OE_TC_131				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**

			#include <string.h>
			
			char *__**strtok**__(char *restrict *s1*, const char *restrict *s2*);
			
			char *__**strtok_r**__(char *restrict *s*, const char *restrict *sep*, char **restrict *lasts*);	

End Of Test				

Test Recording Log – OE_TC_131	
Step	**Functions not present**
Step 1 (options)	
_POSIX_ASYNCHRONOUS_IO	
_POSIX_MEMLOCK_RANGE	
_POSIX_MEMLOCK	
_POSIX_MESSAGE_PASSING	
_POSIX_REALTIME_SIGNALS	
_POSIX_SEMAPHORES	
_POSIX_THREAD_ATTR_STACKADDR	
_POSIX_THREAD_ATTR_STACKSIZE	
_POSIX_THREAD_PRIO_INHERIT	
_POSIX_THREAD_PRIO_PROTECT	
_POSIX_THREAD_PRIORITY_SCHEDULING	

Test Recording Log – OE_TC_131	
Step	**Functions not present**
_POSIX_TIMERS	
Step 5 (Table B-6)	
POSIX_SIGNALS	
Step 8 (Table B-9)	
POSIX_FILE_SYSTEM	
Step 10 (Table B-11)	
POSIX_FD_MGMT	
Step 11 (Table B-12)	
POSIX_DEVICE_IO	
Step 16 (Table B-17)	
POSIX_C_LANG_SUPPORT	
Step 17 (Table B-18)	
POSIX_C_LANG_MATH	
Step 18 (Table B-19)	

Test Recording Log – OE_TC_131	
Step	**Functions not present**
POSIX_C_LANG_JUMP	
Step 19 (Table B-20)	
POSIX_SEMAPHORES	
Step 20 (Table B-21)	
POSIX_TIMERS	
Step 21 (Table B-22)	
POSIX_THREADS_BASE	
Step 22 (Table B-23)	
POSIX_THREADS_SAFE_FUNCTIONS	

Test Summary OE_TC_131

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

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

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

OE0657_____	OE0671_____
OE0658_____	OE0672_____
OE0659_____	OE0673_____
OE0660_____	OE0762_____
OE0663_____	OE0763_____
OE0664_____	OE0764_____
OE0665_____	OE0765_____
OE0667_____	OE0767_____
OE0668_____	OE0768_____
OE0669_____	OE0769_____
OE0670_____	OE0770_____

Failed Items (Section/Step Number):

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.19. OE_TC_155 - Naming Context's execute parameter

Test Case Number: OE_TC_155

ApplicationFactory::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0750	The execute parameter for the Naming Context IOR shall be a CF Properties type with an id element set to "NAMING_CONTEXT_IOR" and a value element set to the stringified IOR of the naming context to which the component will bind.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-28 Section 3.1.3.2.2.5.1.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-22 and C-23

Test Objective

This test case verifies requirement OE0750. There are 3 parts to verify this requirement, of which all 3 parts must be verified and validated to pass this requirement. The first part is to verify the execute parameter for the Naming Context IOR is a CF::Properties type. The second part is the Naming Context IOR is of CF::Properties Type defined with an id element set to "NAMING_CONTEXT_IOR". Lastly, a value element set to the stringified IOR of the naming context.

Note:
The ApplicationFactory interface's *create* operation is used to create an application within the system domain. The *create* operation loads and executes application software modules as specified in the application's Software Assembly Descriptor (SAD) file. The *create* operation includes the mandatory execute parameters as described in SCA section 3.1.3.2.2.5.1.3 (Page 3-28), required for instantiation of the application.

Places to Verify

ApplicationFactory

IDL References

Operations

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

Preconditions

- The ApplicationFactory *create* operation source code files are available.

Test Description

A. Locate and verify the ApplicationFactory *create* operation/method is defined and implemented in the source code. (OE0750)
   1. **Pass:** The ApplicationFactory *create* operation is defined and implemented in the source code.
   2. **Fail:** The ApplicationFactory *create* operation does not exist and/or is NOT fully implemented in the ApplicationFactory source code provided.

B. Within the Naming Context IOR, verify the execute parameter is a CF::Properties type. (OE0750)
   1. **Pass:** The execute parameter for the Naming Context IOR is defined as a CF::Properties type.
   2. **Fail:** The execute parameter for the Naming Context IOR is NOT defined as a CF::Properties type.

C. Verify the id element is set to "NAMING_CONTEXT_IOR". (OE0750)
   1. **Pass:** The id element is set and equal to "NAMING_CONTEXT_IOR".
   2. **Fail:** The id element is NOT set to or equal "NAMING_CONTEXT_IOR".

D. Verify the value element is set to the stringified IOR of the naming context to which the component will bind. (OE0750)
   1. **Pass:** The value element is set to the stringified IOR of the naming context to which the component will bind.
   2. **Fail:** The value element is NOT set to the stringified IOR of the naming context to which the component will bind.

Manual Test Steps

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

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

OE_TC_155				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify the ApplicationFactory *create* operation/method is implemented in the source code. (OE0750)				
1. Identify the source code for the OE where all the ApplicationFactory *create* operation(s) are implemented. Record the source code file(s) names.	**Pass:** The ApplicationFactory *create* operation is implemented in the source code. (OE0750) **Fail:** The ApplicationFactory *create* operation is NOT implemented in the source code. (OE0750)			
B. Verify the execute parameter for the Naming Context IOR is a CF::Properties type. (OE0750)				
2. Within the Naming Context IOR, verify the execute parameter is defined as a CF::Properties type.	**Pass:** The execute parameter for the Naming Context IOR is defined as a CF::Properties type. (OE0750) **Fail:** The execute parameter for the Naming Context IOR is NOT defined as a CF::Properties type. (OE0750)		The execute parameter, the initial configuration, is defined as type CF::Properties. The Properties is a CORBA IDL unbounded sequence of CF::DataType(s), which can be used in defining a sequence of name and value pairs.	
C. Verify the id element is set to "NAMING_CONTEXT_IOR". (OE0750)				
3. Verify the id element is set to "NAMING_CONTEXT_IOR".	**Pass:** The id element is set to "NAMING_CONTEXT_IOR".(OE0750) **Fail:** The id element is NOT set to "NAMING_CONTEXT_IOR".(OE0750)		Search for the string "NAMING_CONTEXT_IOR". id = "NAMING_CONTEXT_IOR"	

OE_TC_155				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			After the ApplicationFactory *create* operation has been called to create an instance of the application the CF::Properties information is used to load and execute the application on the system. The actual code for implementing the setting of the id element may be located in an operation called from the ApplicationFactory::*Create* operation.	
D. Verify the value element is set to the stringified IOR of the naming context to which the component will bind. (OE0750)				
4. Verify the value element is set to the stringified IOR of the naming context to which the component will bind.	**Pass:** The value element is set to the stringified IOR of the naming context to which the component will bind. (OE0750) **Fail:** The value element is NOT set to the stringified IOR of the naming context to which the component will bind. (OE0750)		The actual code for implementing the setting of the value element may be located in an operation called from the ApplicationFactory::*Create* operation.	
End Of Test				

Test Recording Log – OE_TC_155		
Step 1 (source file names that implement the ApplicationFactory::*create* operation)	Step 3 (source file names where the setting of the id element is implemented)	Step 4 (source file names where the setting of the value element is implemented)

Test Summary OE_TC_155

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

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

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

OE0750_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.20. OE_TC_156 - Naming Context creation

Test Case Number: OE_TC_156

DomainManager

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0720	The domain manager shall create a naming context using "**/DomainName**" as the ***id*** attribute to the input name parameter, and "" (Null string) as the ***kind*** attribute.
OE0745	The domain manager shall create a name binding to the created naming context using "**/DomainName**" as the ***id*** attribute to the input name parameter, and "" (Null string) as the ***kind*** attribute, where "DomainName" is identical to the ***name*** attribute of the domain manager's DMD domainmanagerconfiguration element and the input object parameter is the domain manager object reference. [6][1]

References

Index	Document Name	Version/Date	Location (Pages, Section)
1	SCA Specification	Version 2.2.2 15 May 2006	Page 3-35, Section 3.1.3.2.3.5
2	SCA Specification, Appendix D: Domain Profile	Version 2.2.2 15 May 2006	D-54, Section D.8.1
3	OMG Document formal/00-11-01: Interoperable Naming Service Specification	OMG Document formal/00-11-01 November 2000	Page 2-9, Section 2.2.6

Test Objective

This test case verifies SCA v2.2.2 Operating Environment Requirement number OE0720 and OE0745.

The objective of this Test case verifies the DomainManager:

1. creates a naming context, followed by
2. creating a "name binding" to the created naming context in step 1.

The created naming context (step 1) must have the "***id*** value", set equal to, or equivalent to, the name attribute from, and exactly similar to, the domainmanagerconfiguration element and the ***kind*** value set equal to the "Null" string ("").

NOTE: The following list clarifies some interpretations

[1] SCA v2.2.2 reference 6, [6], refers to the OMG Document formal/00-11-01: Interoperable Naming Service Specification specified in the References section of this test cases.

1. The "Null" string is a software programming term, that specifies a variable has, no value, or set to nothing.
2. The domainmanagerconfiguration element is a characteristics found in the DomainManager Configuration Descriptor (DMD) file. This is an SCA v2.2.2 DTD file. The DMD format can be found by referring to Reference [2].

Places to Verify

DomainManager

IDL References

None

Preconditions

- The DomainManager source code files are available.

Test Description

A. Verify the naming context is created, if the ***id*** attribute is equal and set to "**/DomainName**". The Naming context must be created with the following two characteristics:
   - the ***id*** value is, set equal to, or equivalent to, the "***name***" attribute. The ***"name***" attribute in this case is set to "**/DomainName**" The "***name"*** attribute is defined and can be found in the domain manager's DMD file within the "domainmanagerconfiguration" element, and
   - the ***kind*** value is set to Null String ("). (OE0720)

   1. **Pass**: The Domain Manager successfully created the naming context ***id*** when it is set to equal "**/DomainName"**. Additionally, the ***kind*** value is equal to the NULL string.
   2. **Fail**: The Domain Manager was unable to create a naming context ***id*** when it is set to "**/DomainName"**. Additionally, but not dependent upon, the ***kind*** value is also not a NULL string.

B. Verify the domain manager successfully created a "name binding" relative to the created "naming context" listed in Step A. (OE0745)
   - The naming context is created in Step A above. This naming context "value" is used to create the "name binding" used in this step. Therefore, Step A should be successfully prior to passing this step.

   1. **Pass**: The Domain Manager successfully created a "name binding" to the naming context listed in Step A.
   2. **Fail**: The Domain Manager was unable to successfully "name bind(*ing)*" to the naming context listed in Step A.

Manual Test Steps

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

2. The Test Recording Log is intended as a mechanism to record data results, which will be used to solidify and provide verification for test behavior.

OE_TC_156				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that a naming context is created with the id value is set to the "name" attribute of the domain managers DMD domainmanagerconfiguration element and the kind value set to "" (Null String). (OE0720)				
1. Verify where the CORBA Naming Service is created.	CORBA Naming Service exists. **Untested:** The Naming Service is not present, or the Domain Manager is unable to register itself with the Naming Service.		One of the fundamental steps of a JTR is for the Domain Manager to register itself with the naming service. If a Naming service does not exist, the Domain Manager is unable to coexist.	
2. Within the source code, locate the Domain Manager class. Within this class, locate the operation where the Domain Manager registers with the naming service.	Source code is located. **Pass**: Have located the Domain Manager operation where the Domain Manager registers with the Naming Service. **Fail**: Unable to locate the operation where the Domain Manager registers with the Naming Service.		Search for key words "bind" and "bind_new_context" in the Domain Manager source code. Record the name of the CosNaming module NamingContext interface operation bind or bind_new_context that is used to bind the DomainManager to the naming service (or equivalent operation). Record Domain Manager file names examined (in test log).	

OE_TC_156				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. In the provided source code, locate and identify the operation where the naming context is created.	**Pass**: The operation is found where the Naming Context is created. Additionally, a naming context is created. (OE0720) **Fail**: The operation is found where the Naming Context is created, however, a naming context is not created. Alternatively, the software operation is not found for creating the Naming Context. (OE0720)		Search for key word's "new_context", "bind_new_context". Record function used to create naming context (in test log).	
4. Verify in source code the created naming context has an ***id*** value set to "**/DomainName**" of the domain manager's DMD domainmanagerconfiguration element.	**Pass**: The ***id*** value is set to the "**/DomainName**" of the domain managers DMD domainmanagerconfiguration element. (OE0720) **Fail**: The ***id*** value is not set to or equal to, the "**/DomainName**" of the domain managers DMD domainmanagerconfiguration element. (OE0720)		It is recommended to review the XML parsing operation in the source code. Search for key word "CosNaming::Name" <domainmanagerconfiguration id="XXXX" **name**="**YYYY**"> Record applicable variable and file names, in test log.	
5. Verify in source code the created naming context has a ***kind*** value set to "" (Null String).	**Pass**: The ***kind*** value is set to "" (Null String). (OE0720) **Fail**: The ***kind*** value is not set to "" (Null String).(OE0720)		Search for key word "CosNaming::Name" Record function used to create naming context (in test log).	
B. Verify the domain manager successfully created a name binding to the created naming context in the above steps. (OE0752)				

OE_TC_156				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Verify the domain manager successfully created a "*name binding*" to the created naming context (steps listed above). This step is where the JTR is in the startup phase.	**Pass**: From the Naming Context above, a "name binding" is successfully created. (OE0745) **Fail**: From the Naming Context above, a "name binding" is unsuccessfully in being created. (OE0745)		Search for key words "bind_new_context", "CosNaming::NamingContext_var" Record function used to create name binding (in test log).	
End Of Test				

Test Recording Log – OE_TC_156				
Step2 (Locate where the Domain Manager registers with the Naming Service)	**Step3** (source code Function &/or Operation where the Naming Context is created and used)	**Step4** (Verify and validate the ***id*** = "/DomainName")	**Step5** (verify and validate the ***kind*** = NULL)	**Step6** (Within the source code, locate the function &/or operation where the Name Binding is created and used.)

Test Summary OE_TC_156

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

Pass:	No failures detected
Fail:	Failure(s) detected in Step(s) (x).
Untested:	Condition which is not testable
N/A:	Not Applicable

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

OE0720_____

OE0745_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.21. OE_TC_160 - Name Binding's execute parameter

Test Case Number: OE_TC_160

ApplicationFactory::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0751	The Name Binding execute parameter shall be a CF *Properties* type with an id element set to "NAME_BINDING" and a value element set to a string in the format of "ComponentName_UniqueIdentifier".

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-26 to 3-31, Section 3.1.3.2.2.2
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-23

Test Objective

This test case verifies OE0751. The objective of this test is to verify that the Name Binding execute parameter is a CF::*Properties* type with an id element set to "NAME_BINDING" and a value element set to a string in the format of "ComponentName_UniqueIdentifier".

Places to Verify

The code that implements the ApplicationFactory create method.

IDL References

Data

typedef sequence <DataType> Properties;

Operations

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

```
        CF::ApplicationFactory::InvalidInitConfiguration);
};
```

Preconditions

- The *ApplicationFactory* source code files are available.

Test Description

A. Locate the source code, in the implementation of ApplicationFactory create, that gets the Name Binding that will be passed as a parameter to the ExecutableDevice::execute operation call. (OE0751)
   1. **Pass:** The source code for the Name Binding execute parameter that will be passed to the ExecutableDevice::execute operation call **is** located.
   2. **Untested**: The source code for the Name Binding execute parameter that will be passed to the ExecutableDevice::execute operation call is **not** located.

B. Verify that the id element is set to "NAME_BINDING". (OE0751)
   1. **Pass:** The id element is set to "NAME_BINDING".
   2. **Fail:** The id element is not set to "NAME_BINDING".

C. Verify that the Name Binding execute parameter is a CF::Properties type. (OE0751)
   1. **Pass:** The Name Binding execute parameter **is** a CF::Properties type.
   2. **Fail:** The Name Binding execute parameter is **not** a CF::Properties type.

D. Verify that the value element is set to a string in the format of "ComponentName_UniqueIdentifier". (OE0751)
   1. **Pass:** The Name Binding execute parameter value element **is** set to a string in the format of "ComponentName_UniqueIdentifier".
   2. **Fail:** The Name Binding execute parameter value element is **not** set to a string in the format of "ComponentName_UniqueIdentifier".

Manual Test Steps

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

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

OE_TC_160				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the Name Binding execute parameter that will be passed to the ExecutableDevice::execute operation call. (OE0751)				
1. Locate the source code, in the implementation of ApplicationFactory create, that gets the Name Binding that will be passed as a parameter to the ExecutableDevice::execute operation call. Performing a key word search on all source code provided by the developer for "NAME _BINDING". Record source code file name.	**Pass:** The source code for the Name Binding execute parameter that will be passed to the ExecutableDevice::execute operation call **is** located. (OE0751) **Untested**: The source code for the Name Binding execute parameter that will be passed to the ExecutableDevice::execute operation call is **not** located. (OE0751)			
B. Verify that the id element is set to "NAME_BINDING". (OE0751)				
2. From the source code file recorded in the previous step verify that the id element is set to "NAME_BINDING".	**Pass:** The id element **is** set to "NAME_BINDING". (OE0751) **Fail:** The id element is **not** set to "NAME_BINDING". (OE0751)		This will likely be located with the code building all the necessary parameters to the execute method.	
C. Verify that the Name Binding execute parameter is a CF::Properties type. (OE0751)				
3. From the source code file recorded in the step 1 verify that the Name Binding execute parameter is a CF::Properties type.	**Pass:** The Name Binding execute parameter **is** a CF::Properties type. (OE0751) **Fail:** The Name Binding execute parameter is **not** a CF::Properties type. (OE0751)		Parameters passed with the execute command.	
D. Verify that the Name Binding execute parameter is a value element set to a string in the format of "ComponentName_UniqueIdentifier". (OE0751)				

OE_TC_160				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. From the source code file recorded in the step 1 verify that the Name Binding execute parameter is a value element set to a string in the format of "ComponentName_UniqueIdentifier".	**Pass:** The Name Binding execute parameter value element **is** set to a string in the format of "ComponentName_UniqueIdentifier". (OE0751) **Fail:** The Name Binding execute parameter value element is **not** set to a string in the format of "ComponentName_UniqueIdentifier". (OE0751)		The UniqueIdentifier is determined by the implementation.	
End Of Test				

Test Recording Log – OE_TC_160				
Step1 (source code file name)	**Step2** (is set to "NAME_BINDING) Y/N	**Step3** (is a CF::Properties type) Y/N	**Step4** (formatted "ComponentName_UniqueIdentifier") Y/N	**Notes**

Test Summary OE_TC_160

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

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

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

OE0751 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.22. OE_TC_161 - Component Identifier's execute parameter

Test Case Number: OE_TC_161

ApplicationFactory::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0752	The Component Identifier execute parameter shall be a CF Properties type with an id element set to "COMPONENT_IDENTIFIER" and a value element set to a string in the format of "Component_Instantiation_Identifier: Application_Name".

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-28, 3-29, Section 3.1.3.2.2.5.1.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-22 and C-23

Test Objective

This test case verifies requirement OE0752. The objective of this test is to verify that the execute parameter for the Component Identifier is a CF Properties type with an id element set to "COMPONENT_IDENTIFIER" and a value element set to a string in the format of "Component_Instantiation_Identifier: Application_Name".

Note:
The ApplicationFactory interface's *create* operation is used to create an application within the system domain. The *create* operation loads and executes application software modules as specified in the application's Software Assembly Descriptor (SAD) file. The *create* operation includes the mandatory execute parameters as described in SCA section 3.1.3.2.2.5.1.3, required for instantiation of the application.

Places to Verify

ApplicationFactory

IDL References

Operations

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

CF::ExecutableDevice::ProcessID_Type execute (
         in string name,
         in CF::Properties options,
         in CF::Properties parameters )

         raises (CF::Device::InvalidState,
         CF::ExecutableDevice::InvalidFunction,
         CF::ExecutableDevice::InvalidParameters,
         CF::ExecutableDevice::InvalidOptions,
         CF::InvalidFileName,
         CF::ExecutableDevice::ExecuteFail); };
```

Preconditions

- The ApplicationFactory create operation source code files are available.

Test Description

NOTE: If this test is being executed because JTAP failed to verify the requirement OE0752, please note that OE0752-C140, which is part of OE0752, is not currently verified.

A. Verify that the ApplicationFactory *create* operation is implemented in the source code. (OE0752)

1. **Pass:** The ApplicationFactory *create* operation is implemented in the source code.
2. **Fail:** The ApplicationFactory *create* operation is NOT implemented in the source code.

B. Verify that the execute parameter for the Component Identifier is a CF Properties type. (OE0752)
    1. **Pass**: The execute parameter for the Component Identifier is a CF Properties type.
    2. **Fail**: The execute parameter for the Component Identifier is NOT a CF Properties type.
C. Verify that the id element is set to "COMPONENT_IDENTIFIER". (OE0752)
    a. **Pass**: The id element is set to "COMPONENT_IDENTIFIER".
    b. **Fail**: The id element is NOT set to "COMPONENT_IDENTIFIER".
D. Verify that the value element is set to a string in the format of "Component_Instantiation_Identifier: Application_Name". (OE0752)
    1. **Pass**: The value element is set a string in the format of "Component_Instantiation_Identifier: Application_Name".
    2. **Fail**: The value element is NOT set to the string in the format of "Component_Instantiation_Identifier: Application_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.

OE_TC_161				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the ApplicationFactory *create* operation is implemented in the source code.				
1. Identify the source code for the OE where all the ApplicationFactory *create* operation(s) are implemented. Record the source code file(s) names.	**Pass:** The ApplicationFactory create operation is implemented in the source code. (OE0752) **Fail:** The ApplicationFactory create operation is NOT implemented in the source code. (OE0752)		The *create()* operation is responsible for the *execute()* operation being called. The *create()* operation may not directly call the *execute()* operation.	
B. Verify that the execute parameter for the Component Identifier is a CF Properties type.				
2. From the source files recorded in the previous step verify that the execute parameter for the Component Identifier is a CF Properties type. Record the CF::Properties variable name in test log.	**Pass:** The execute parameter for the Component Identifier is a CF Properties type. (OE0752) **Fail:** The execute parameter for the Component Identifier is NOT a CF Properties type. (OE0752)		*execute(* *in string name,* *in CF::Properties options,* *in CF::Properties parameters* *)* The execute() operation has 3 parameters. The parameter that is being verified in this test case is labeled "*parameters"* in the above IDL code. The execute parameter, the initial configuration passed into the create operation is of type CF::Properties.	

OE_TC_161				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
			The Properties is a CORBA IDL unbounded sequence of CF DataType(s), which can be used in defining a sequence of name and value pairs.	
C. Verify that the id element is set to "COMPONENT_IDENTIFIER".				
3. Verify that the id element is set to "COMPONENT_IDENTIFIER".	**Pass:** The id element is set to "COMPONENT_IDENTIFIER".(OE0752) **Fail:** The id element is NOT set to "COMPONENT_IDENTIFIER".(OE0752)		Search for the string "COMPONENT_IDENTIFIER". After the ApplicationFactory create operation has been called to create an instance of the application, the CF::Properties id element is set to "COMPONENT_IDENTIFIER". Record the name of the CF::Properties variable and the value the id element is set to, in test log.	
D. Verify that the value element is set to a string in the format of "Component_Instantiation_Identifier: Application_Name"				
4. Verify that the value element is set to a string in the format of "Component_Instantiation_Identifier: Application_Name".	**Pass:** The value element is set to a string in the format of "Component_Instantiation_Identifier: Application_Name".(OE0752) **Fail:** The value element is NOT set to a string in the format of "Component_Instantiation_Identifier: Application_Name". (OE0752)		array[i].value<<=compID.str(); Record the name of the CF::Properties variable and the value the value element is set to, in test log.	
End Of Test				

Test Recording Log – OE_TC_161			
Step 1 (*create* operation source file names)	**Step 2** (variable name)	**Step 3** (verify id element value)	**Step 4** (verify format of value element value)

Test Summary OE_TC_161

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

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

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

OE0752_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.5.23. OE_TC_290 - Networking AEP

Test Case Number: OE_TC_290

OS::Networking AEP

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0819	The functions listed in Table 1-2 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table 1-1.
OE0820	The functions listed in Table 1-3 shall behave as described in the applicable clauses of the referenced POSIX specifications contained in Table 1-1.

References

Document Name	Version/Date	Location (Pages, Section)
SCA Networking Application Environment Profile	Version 2.2.2A 19 March 2010	Page 7, Section 1.4.1.1; Page 8, Section 1.4.1.2
The Open Group Base Specifications Issue 6, IEEE Std 1003.1	2004 Edition	http://www.opengroup.org/onlinepubs/000095399/

Test Objective

This test case verifies OE0819 and OE0820. The objective of this test is to verify that the OE supplies the functions designated as mandatory in SCA Networking Application Environment Profile. According to the direction of JTNC Standards, it is not considered as failure when the OE cannot provide any or all of the mandatory functions listed in Table 1-2 and 1-3 in the SCA Networking Application Environment Profile.

Places to Verify

OE header files, including arpa/inet.h, sys/socket.h, and sys/select.h.

IDL References

None.

Preconditions

- The OE header source code files are available.

Test Description

A. Verify that the mandatory functions from Table 1-2 in SCA Networking AEP are provided. (OE0819)
    1. **Pass**: The mandatory functions are provided.
    2. **N/A**: All of the mandatory functions are not provided.
B. Verify that the mandatory functions from Table 1-3 in SCA Networking AEP are provided. (OE0820)
    1. **Pass**: The mandatory functions are provided.
    2. **N/A**: All of the mandatory functions are not provided.

Manual Test Steps

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

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

OE_TC_290				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the mandatory functions from Table 1-2 are provided. (OE0819)				
1. Verify that the mandatory functions from Table 1-2 in SCA Networking AEP are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0819) **N/A**: None or not all of the mandatory functions are provided. (OE0819)		The mandatory functions from Table 1-2 are: (from sys/socket.h) accept ( ) bind ( ) connect ( ) getsockopt ( ) listen ( ) recv ( ) recvfrom ( ) send ( ) sendto ( ) setsockopt ( ) socket ( ) (from arpa/inet.h) htonl ( ) htons ( ) ntohl ( ) ntohs ( )	
B. Verify that the mandatory functions from Table 1-3 are provided. (OE0820)				
2. Verify that the mandatory functions from Table 1-3 in SCA Networking AEP are provided in the OE header files.	**Pass**: The mandatory functions are provided. (OE0820) **N/A**: None or not all of the mandatory functions are provided. (OE0820)		The only mandatory function from Table 1-3 is: (from sys/select.h) Select()	

OE_TC_290				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
End Of Test				

Test Recording Log – OE_TC_290	
Step 1 (Table 1-2)	**Functions not present**
(from sys/socket.h) accept ( ) bind ( ) connect ( ) getsockopt ( ) listen ( ) recv ( ) recvfrom ( ) send ( ) sendto ( ) setsockopt ( ) socket ( ) (from arpa/inet.h) htonl ( ) htons ( ) ntohl ( ) ntohs ( )	
Step 2 (Table 1-3)	**Functions not present**
(from sys/select.h) Select()	

Test Summary OE_TC_290

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

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

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

OE0819_____

OE0820_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

Index of Test Case Titles for Manual Tests

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
AEP Applications have no abnormal termination	5	31	B.5.4.	OE_TC_021
AEP file mode creation masks	5	37	B.5.5.	OE_TC_044
AEP mandatory functions	5	142	B.5.18.	OE_TC_131
Aggregate Device	3	191	B.3.25.	OE_TC_096
AggregateDevice devices	3	12	B.3.2.	OE_TC_022
AggregateDevice :: addDevice	3	20	B.3.3.	OE_TC_023
AggregateDevice :: addDevice FAILURE_ALARM	3	29	B.3.4.	OE_TC_025
AggregateDevice :: addDevice raises InvalidObjectReference	3	35	B.3.5.	OE_TC_027
AggregateDevice :: removeDevice	3	42	B.3.6.	OE_TC_028
AggregateDevice :: removeDevice FAILURE_ALARM	3	48	B.3.7.	OE_TC_029
AggregateDevice :: removeDevice raises InvalidObjectReference	3	54	B.3.8.	OE_TC_030
Application :: releaseObject releases all objects	2	96	B.2.16.	OE_TC_108
Application Attributes	2	258	B.2.39.	OE_TC_233
Application Delegates Implementation of Resource operations	2	237	B.2.38.	OE_TC_189
ApplicationFactory :: create	2	84	B.2.14.	OE_TC_074
ApplicationFactory :: create deallocates capacity on devices	2	201	B.2.32.	OE_TC_162
ApplicationFactory :: create defines an order for initialization	2	212	B.2.34.	OE_TC_165
ApplicationFactory :: create does property comparisons	2	119	B.2.20.	OE_TC_121
ApplicationFactory :: create establishes connections for named applications	2	207	B.2.33.	OE_TC_164
ApplicationFactory :: create raises CreateApplicationError	2	138	B.2.23.	OE_TC_125
ApplicationFactory Attributes	2	258	B.2.41.	OE_TC_236
Base Application Interfaces	1	263	B.1.35.	OE_TC_102
Base Device Interfaces	3	71	B.3.10.	OE_TC_062
ComponentIdentifier's execute parameter	5	200	B.5.22.	OE_TC_161
CORBA :: CosEventComm	5	52	B.5.8.	OE_TC_063
CORBA :: Log Producer	5	15	B.5.2.	OE_TC_002

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
CORBA :: NamingService	5	5	B.5.1.	OE_TC_001
Device :: adminState attribute changes	3	154	B.3.20.	OE_TC_088
Device :: adminState commanded to be LOCKED	3	91	B.3.12.	OE_TC_080
Device :: allocateCapacity	3	6	B.3.1.	OE_TC_004
Device :: allocateCapacity	3	105	B.3.13.	OE_TC_081
Device :: allocateCapacity BUSY	3	114	B.3.14.	OE_TC_082
Device :: allocateCapacity Failure	3	122	B.3.15.	OE_TC_083
Device :: allocateCapacity raises exceptions	3	299	B.3.39.	OE_TC_229
Device :: allocateCapacity's acceptable properties	3	217	B.3.28.	OE_TC_118
Device :: deallocateCapacity	3	128	B.3.16.	OE_TC_084
Device :: deallocateCapacity raises InvalidCapacity	3	141	B.3.18.	OE_TC_086
Device :: deallocateCapacity raises InvalidState	3	305	B.3.40.	OE_TC_230
Device :: deallocateCapacity usageState	3	134	B.3.17.	OE_TC_085
Device :: Logical Device - CORBA	3	177	B.3.23.	OE_TC_094
Device :: Logical Device - CORBA Register	3	184	B.3.24.	OE_TC_095
Device :: Logical Device allocation properties	3	206	B.3.27.	OE_TC_098
Device :: Logical Device executable parameters	3	168	B.3.22.	OE_TC_092
Device :: Logical Devices - CF Interfaces	3	198	B.3.26.	OE_TC_097
Device :: releaseObject	3	147	B.3.19.	OE_TC_087
Device :: releaseObject raises ReleaseError exception	3	162	B.3.21.	OE_TC_089
Device :: usageState	3	77	B.3.11.	OE_TC_079
Device Attributes	3	261	B.3.35.	OE_TC_218
Device operationalState	3	60	B.3.9.	OE_TC_058
DeviceManager :: getComponentImplementationId	2	89	B.2.15.	OE_TC_090
DeviceManager Attributes	2	247	B.2.40.	OE_TC_224
DeviceManager Register	2	17	B.2.3.	OE_TC_033
DeviceManager Register and the DCD file	2	23	B.2.4.	OE_TC_034
DeviceManager startup process	2	277	B.2.42.	OE_TC_285

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
DeviceManager's execparamproperties	2	101	B.2.17.	OE_TC_112
Domain Manager logs defined in DMD file	2	49	B.2.8.	OE_TC_054
Domain Manager services defined in DMD file	2	61	B.2.10.	OE_TC_056
Domain Profile	5	58	B.5.9.	OE_TC_070
Domain Profile files	5	114	B.5.17.	OE_TC_119
DomainManager :: installApplication raises ApplicationAlreadyInstalled	2	11	B.2.2.	OE_TC_026
DomainManager :: installApplication raises ApplicationInstallationError	2	67	B.2.11.	OE_TC_057
DomainManager :: installApplication raises InvalidFileName	2	172	B.2.28.	OE_TC_132
DomainManager :: registerDevice	2	107	B.2.18.	OE_TC_113
DomainManager :: registerDevice raises RegisterError	2	132	B.2.22.	OE_TC_124
DomainManager :: registerDevice registers if device doesn't exist	2	230	B.2.37.	OE_TC_168
DomainManager :: registerDevice returns without error if device exists	2	224	B.2.36.	OE_TC_167
DomainManager :: registerDevice verifies input parameters	2	218	B.2.35.	OE_TC_166
DomainManager :: registerDeviceManager	2	6	B.2.1.	OE_TC_024
DomainManager :: registerDeviceManager establishes connections	2	188	B.2.30.	OE_TC_157
DomainManager :: registerDeviceManager raises RegisterError	2	146	B.2.24.	OE_TC_126
DomainManager :: registerDeviceManager sends event to ODM	2	180	B.2.29.	OE_TC_154
DomainManager :: registerService	2	73	B.2.12.	OE_TC_065
DomainManager :: registerService	2	79	B.2.13.	OE_TC_073
DomainManager :: registerService raises RegisterError	2	153	B.2.25.	OE_TC_127
DomainManager :: uninstallApplication raises ApplicationUninstallationError	2	125	B.2.21.	OE_TC_122
DomainManager :: unregisterDevice raises UnregisterError	2	166	B.2.27.	OE_TC_129
DomainManager :: unregisterDeviceManager	2	43	B.2.7.	OE_TC_053
DomainManager :: unregisterDeviceManager	2	194	B.2.31.	OE_TC_158
DomainManager :: unregisterDeviceManager raises UnregisterError	2	159	B.2.26.	OE_TC_128
DomainManager :: unregisterService	2	32	B.2.5.	OE_TC_051
DomainManager :: unregisterService client-side	2	38	B.2.6.	OE_TC_052
DomainManager :: unregisterService raises UnregisterError	2	55	B.2.9.	OE_TC_055

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
DTD files	5	108	B.5.16.	OE_TC_117
Event Service	5	23	B.5.3.	OE_TC_003
Exceptions from the Mounted FileSystem	4	18	B.4.2.	OE_TC_040
ExecutableDevice :: execute	3	290	B.3.38.	OE_TC_227
ExecutableDevice :: execute raises exceptions	3	278	B.3.37.	OE_TC_222
ExecutableDevice :: execute raises InvalidFileName	3	244	B.3.32.	OE_TC_135
ExecutableDevice :: terminate raises InvalidProcess	3	250	B.3.33.	OE_TC_145
ExecutableDevice Types	3	272	B.3.36.	OE_TC_221
File :: close	4	55	B.4.5.	OE_TC_067
File :: close raises FileException	4	60	B.4.6.	OE_TC_068
File :: read raises IOException	4	49	B.4.4.	OE_TC_066
File :: setFilePointer raises FileException	4	66	B.4.7.	OE_TC_069
File :: sizeOf raises FileException	4	215	B.4.29.	OE_TC_147
File :: write raises IOException	4	208	B.4.28.	OE_TC_146
File Attributes	4	262	B.4.36.	OE_TC_265
FileManager :: list	4	6	B.4.1.	OE_TC_031
FileManager :: mount	4	278	B.4.38.	OE_TC_276
FileManager :: mount raises InvalidFileName	4	194	B.4.26.	OE_TC_143
FileSystem:: copy	4	99	B.4.12.	OE_TC_106
FileSystem:: copy raises InvalidFileName when inputs are invalid	4	148	B.4.20.	OE_TC_137
FileSystem:: copy raises InvalidFileName when inputs are the same	4	202	B.4.27.	OE_TC_144
FileSystem:: create	4	106	B.4.13.	OE_TC_107
FileSystem:: create raises FileException	4	227	B.4.31.	OE_TC_149
FileSystem:: create raises InvalidFileName	4	165	B.4.22.	OE_TC_139
FileSystem:: exists	4	159	B.4.21.	OE_TC_138
FileSystem:: exists raises InvalidFileName	4	254	B.4.35.	OE_TC_159
FileSystem:: list raises FileException	4	87	B.4.10.	OE_TC_104
FileSystem:: list raises InvalidFileName	4	78	B.4.9.	OE_TC_103

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
FileSystem:: mkdir	4	129	B.4.17.	OE_TC_114
FileSystem:: mkdir raises FileException	4	241	B.4.33.	OE_TC_151
FileSystem:: mkdir raises InvalidFileName	4	179	B.4.24.	OE_TC_141
FileSystem:: open raises FileException	4	233	B.4.32.	OE_TC_150
FileSystem:: open raises InvalidFileName	4	172	B.4.23.	OE_TC_140
FileSystem:: query Input Parameter	4	269	B.4.37.	OE_TC_275
FileSystem:: remove raises FileException	4	93	B.4.11.	OE_TC_105
FileSystem:: remove raises InvalidFileName	4	141	B.4.19.	OE_TC_136
FileSystem:: rmdir	4	135	B.4.18.	OE_TC_115
FileSystem:: rmdir raises FileException	4	247	B.4.34.	OE_TC_152
FileSystem:: rmdir raises InvalidFileName	4	187	B.4.25.	OE_TC_142
FileSystem create Responsibilities	4	286	B.4.39.	OE_TC_283
FileSystem Filename Lengths	4	38	B.4.3.	OE_TC_060
FileSystem::copy raises FileException	4	221	B.4.30.	OE_TC_148
FileSystem's CREATED_TIME_ID property	4	111	B.4.14.	OE_TC_109
FileSystem's LAST_ACCESS_TIME_ID property	4	123	B.4.16.	OE_TC_111
FileSystem's MODIFIED_TIME_ID property	4	117	B.4.15.	OE_TC_110
Framework Control Interfaces	2	112	B.2.19.	OE_TC_116
Framework Services Interfaces	4	72	B.4.8.	OE_TC_091
General Rules : Higher Order Language	5	90	B.5.13.	OE_TC_099
Hardware Critical Interfaces	5	103	B.5.15.	OE_TC_101
Legacy Software interfaces	5	95	B.5.14.	OE_TC_100
LifeCycle :: initialize raises InitializeError	1	168	B.1.19.	OE_TC_037
LifeCycle :: releaseObject	1	173	B.1.20.	OE_TC_038
LifeCycle :: releaseObject raises ReleaseError	1	179	B.1.21.	OE_TC_039
LoadableDevice :: load	3	256	B.3.34.	OE_TC_153
LoadableDevice :: load raises InvalidFileName	3	230	B.3.30.	OE_TC_133
LoadableDevice :: load raises LoadFail	3	224	B.3.29.	OE_TC_130

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
LoadableDevice :: unload raises InvalidFileName	3	237	B.3.31.	OE_TC_134
Mandatory interfaces of the AEP	5	43	B.5.6.	OE_TC_059
Minimum CORBA	5	47	B.5.7.	OE_TC_061
Name Binding's execute parameter	5	194	B.5.21.	OE_TC_160
Naming Context creation	5	186	B.5.20.	OE_TC_156
Naming Context's execute parameter	5	180	B.5.19.	OE_TC_155
Networking AEP	5	208	B.5.23.	OE_TC_290
OMGLightweightLogService :: administrative_state	1	37	B.1.5.	OE_TC_009
OMGLightweightLogService :: clear_log	1	131	B.1.14.	OE_TC_018
OMGLightweightLogService :: destroy	1	139	B.1.15.	OE_TC_019
OMGLightweightLogService :: get_availability_status	1	31	B.1.4.	OE_TC_008
OMGLightweightLogService :: get_n_records	1	20	B.1.2.	OE_TC_006
OMGLightweightLogService :: get_operational_state	1	45	B.1.6.	OE_TC_010
OMGLightweightLogService :: get_record_id_from_time	1	51	B.1.7.	OE_TC_011
OMGLightweightLogService :: LogFullAction	1	25	B.1.3.	OE_TC_007
OMGLightweightLogService :: LogStatus	1	8	B.1.1.	OE_TC_005
OMGLightweightLogService :: retrieve_records	1	56	B.1.8.	OE_TC_012
OMGLightweightLogService :: retrieve_records_by_level	1	67	B.1.9.	OE_TC_013
OMGLightweightLogService :: retrieve_records_by_producer_id	1	78	B.1.10.	OE_TC_014
OMGLightweightLogService :: retrieve_records_by_producer_name	1	91	B.1.11.	OE_TC_015
OMGLightweightLogService :: write_record	1	117	B.1.13.	OE_TC_017
OMGLightweightLogService :: write_records	1	103	B.1.12.	OE_TC_016
Port :: connectPort	1	147	B.1.16.	OE_TC_020
Port :: connectPort raises OccupiedPort	1	155	B.1.17.	OE_TC_032
Port :: disconnectPort	1	161	B.1.18.	OE_TC_035
PropertySet :: configure	1	212	B.1.27.	OE_TC_047
PropertySet :: configure property minimums	1	218	B.1.28.	OE_TC_048
PropertySet :: configure raises PartialConfiguration	1	225	B.1.29.	OE_TC_049

Test Case Title	App B Volume #	Page	Section Number	Test Case Number
Provided interfaces described as provides ports	5	67	B.5.10.	OE_TC_075
Required interfaces described as uses ports	5	73	B.5.11.	OE_TC_076
Resource :: start raises StartError	1	231	B.1.30.	OE_TC_050
Resource :: stop	1	241	B.1.32.	OE_TC_071
Resource :: stop raises StopError	1	236	B.1.31.	OE_TC_064
SCA APIs and non-IDL interfaces	5	79	B.5.12.	OE_TC_077
TestableObject :: runTest exception parameter	1	257	B.1.34.	OE_TC_093
TestableObject :: runTest parameters	1	246	B.1.33.	OE_TC_072
TestableObject :: runTest returns results	1	195	B.1.24.	OE_TC_043
TestableObject :: runTest uses testId parameter	1	184	B.1.22.	OE_TC_041
TestableObject :: runTest uses testValues parameter	1	189	B.1.23.	OE_TC_042
TestableObject :: runTest validates parameters	1	206	B.1.26.	OE_TC_046
TestableObject :: runTest validates testValues parameter	1	201	B.1.25.	OE_TC_045
End of Table				