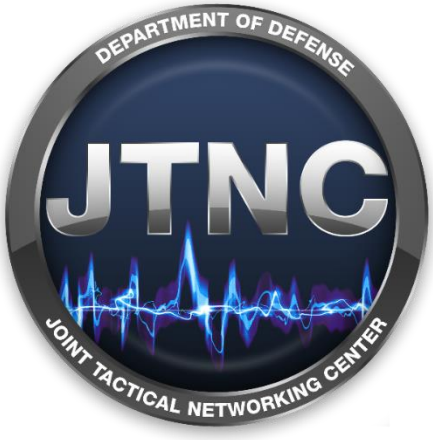
Joint Tactical Networking Center
Test and Evaluation Laboratory

Software Communications Architecture v2.2.2
Manual Application Software Test Description

Version 2.3A

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

Change History

Version	Date Modified	Author	Reason for Change
2.3	27 September 2019	JTEL	Initial release
2.3A	13 April 2022	JTEL	Public Release

Table of Contents

LIST OF TABLES

1 Introduction

1.1 Intended Readership

This document is intended for groups involved with Joint Tactical Radio (JTR) Set Development with Joint Tactical Networking Center (JTNC) Test and Evaluation (JTEL) responsibility including JTR Applications Developers, Applications Test Personnel, JTR Set Manufacturers, JTR Set Integrators, and Joint Tactical Networking Center (JTNC). This document is written assuming that the readers are familiar with the objectives and requirements of the JTNC organization.

1.2 Purpose

The purpose of this document is to provide the applications-specific test procedures to perform the verification of the SCA v2.2.2 application requirements. The application is designed to meet the requirements and interface with the Operating Environment. The results of the tests will provide the information needed to determine compliance to the SCA v2.2.2 [Reference 3] which will be used to make the recommendation to the JTNC. The JTEL SCA Test and Evaluation Plan [Reference 7] provides more information on the compliance recommendation process and the overall testing methods. Identification of the test cases, their objectives/purposes and the relevant SCA v2.2.2 requirements are described in Section 4.2.3.

1.3 JTEL Overview

The JTNC Test and Evaluation Laboratory (JTEL) was established under JTNC to act as the Test Authority (TA) for compliance testing of the SCA. The software defined radio systems are based on applications and platform solutions which share a common architecture as defined by the SCA specification.

Adherence to the SCA allows standardization of the JTR Set infrastructure. Interoperability will be achieved through the reuse of common core waveforms used in all JTR Sets, and through an expanded Defense Information Systems Agency (DISA) Joint Interoperability Test Command (JITC) waveform conformance testing process as a prerequisite to interoperability testing and certification.

JTEL's primary functions include the following:

Provide development and test support for application and Operating Environment testing for the SCA Compliance Certification.

Update and maintain test capabilities, including test tools and test procedure documents.

Remain current with the SCA as it evolves.

Provide support and feedback to JTNC Standards and JTEL customers on requirements interpretations and testability.

Provide compliance test reports and recommendations on the standards compliance of the tested products to the JTEL for certification consideration. Some standards JTEL supports are the SCA v2.2, the SCA v2.2.2 and the JTNC Standard APIs.

Support all US DOD JTR manufacturers that request an independent analysis of their JTR, providing fee-for-service support for-JTR developers.

1.4 Scope

The applications compliance testing is executed by either the manual test procedures, the JTAP tool, or in some circumstances, a combination of both. The scope of this document is to describe the manual test procedures exercised by JTEL in testing for SCA compliance.

Manual testing can be done in varying degrees. For purposes of describing and delineating the test and evaluation documents, the testing types are described here. The extent and range of manual testing can be expressed in the following ways:

Manual testing – This is where artifacts, such as documents and source code, can be visually inspected and analyzed. This type of fundamental assessment is the purest form of manual testing.

Semi-automated testing – As much of the verification of a requirement is performed using JTAP. The portions of a requirement that cannot be verified using JTAP; will be verified using documentation and source code of the developer.

Automated testing – This testing is where the JTAP tool is responsible for verifying requirements and reporting on their success and failure. Because this portion of the testing would be controlled by the JTAP tool, the test procedures for this type of testing will be addressed in the JTAP document [Reference 1].

The descriptions of the automated tests are not in the scope of this Manual Application Standard Test Description (MASTD) document. The names of relevant JTAP automated tests can be found in Appendix B.

1.5 Associated Test Documents

If the JTAP tool can partially or fully automate the test, the JTAP automated applications test document will further describe the applicable automated test [References 1 and 2].

The parent document of this MASTD document, the JTEL SCA Test and Evaluation Plan [Reference 7], provides the high-level information to this document and the other JTEL Test and Evaluation (T & E) documents.

1.6 Document Overview

Section 1: Introduction
This section contains the introduction to the document, the intended audience, the identification and scope of the document, and also provides the high-level description of JTNC and JTEL. It also provides identification of the system, and the software to which this document applies. The system and software information includes, as applicable, identification number(s), title(s), version number(s) and release number(s).

Section 2: Reference Documents
This section contains the documents that are either referenced and/or are used in preparation of this document. Each of the referenced and related documents are described by a title and identifying number(s).

Section 3: Test Environment
This section describes the environments and the configurations used for testing JTR applications for SCA compliance.

Section 4: Test Description

This section contains an introduction to the application-related manual test procedures. This introduction includes identifying the naming conventions for the test names, the explanation of the SCA requirement tagging abbreviations and the objectives of each of the test procedures. This section refers to Appendix B which contains the detailed step-by-step manual test procedures.

Section 5: Requirements Traceability
This section contains information about application requirements and mapping to their test cases. It also describes the requirements identification tag used for identifying the SCA requirements in test cases and the requirements list.

Appendices
Appendix A contains the abbreviations found in this document.

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

Appendix C contains the test case procedures for the AEP Amended requirements.

Appendix D contains the test case procedures for the SCA Extension requirements.

2 Reference Documents

The following documents are the standards, specifications and handbooks that are referenced in or related to this MASTD document.

Table 1: Referenced or Related Documents

Reference#	Document Title
1	JTNC Test and Evaluation Laboratory(JTEL) SCA v2.2.2 Automated Applications Software Test Description(AASTD) v3.14A, 13 April 2022
2	JTNC Test and Evaluation Laboratory(JTEL) JTNC Test Application(JTAP) Software User Manual (SUM) for SCA v2.2.2, v3.14A, 13 April 2022
3	SCA v2.2.2 Specification 15 May 2006
4	SCA v2.2.2 Specification Appendix B: Application Environment Profile 15 May 2006
5	SCA v2.2.2 Specification Appendix C: Core Framework IDL 15 May 2006
6	SCA v2.2.2 Specification Appendix D: Domain Profile 15 May 2006
7	SCA v2.2.2 Test and Evaluation Plan 13 April 2022

3 Test Environment

The three types of environments are Manual only, Semi-automated and Automated testing.

A Manual only test environment uses a rudimentary environment involving visual inspections of source code and other files, such as XML files.

A Semi-Automated test environment consists of visual inspections and JTAP to assist testing.

An Automated test environment uses only the JTAP tool. [See References 1 and 2] for JTAP test environment descriptions.]

3.1 Hardware/Software Preparations

The hardware and software preparations needed to test the requirements should be completed prior to the start of the testing. The arrangements will be tailored by JTEL according to the test event resources and the extent that the tests are manually tested or tested with JTAP. The following items can be used to perform visual inspections of source code and documents:

Printer to print out desired hard copies of test documents, referenced specifications, etc.

A separate host platform capable of supporting online viewing of files for inspection.

A text editor for online viewing of software files and writing test results.

Text processing script application to parse strings in files.

Compiler for the languages in which the application source code is written to check for dependencies and check for proper syntax.

XML edit tools to view XML formatted files.

File comparison tools.

3.2 Test Preparation

The following testware, supporting test documents and preparation steps are recommended to be available and/or performed by the tester prior to the beginning of testing.

List of any waivers for requirements.

Copies of the Test Cases with procedure steps, in a printed or an electronic form.

Test supporting documents identified in Section 2, such as:

SCA v2.2.2 Specification [Reference 3] and appendices

minimumCORBA Specification

OMG Lightweight Log Service specification

Test Recording Log sheets are provided near the end of each test case. These Log sheets allow a tester to write down intermediate test results during the execution of the test. Each Log Sheet is test case-specific. If hand-writing the test results, provide additional copies of the Log Sheet.

Review the items in the Preconditions section of each test case. Preconditions are items that need to be completed or be available before the test has started. For example, source code needs to be available or XML files need to be properly formatted prior to the time of running the specific test.

4 Test Description

4.1 General Information

This section provides general information applicable to the overall testing to be performed.

4.2 Planned Tests

The planned tests that will verify the SCA v2.2.2 compliance are listed in the following sections. Requirements are logically grouped together into test cases, when appropriate. Each test case name follows the naming convention described in the next section.

4.2.1 Test Case Naming Conventions

The general naming format of the test cases is in the format of 'APP_TC_xxx' and is shown in Table 2 below:

Table 2: Test Case Naming Convention

Abbreviations	Meaning
APP	Application
TC	Test Case
xxx	Unique 3-digit numerical identifier of test case

4.2.2 SCA v2.2.2 Application Requirements

The SCA v2.2.2 application requirements that are being tested are contained in Table 5 seen in Section 5. The table identifies the manual test cases. Each test case is associated with the requirement's identifier, the text of the requirement and additional information describing the requirement. The next section provides the list of the manual test cases and additional descriptions about each test case.

4.2.3 SCA v2.2.2 Manual Test Cases and Test Objectives – Summary

The following subsections list the application test cases, the functional area(s), and objectives.

4.2.3.1 APP_TC_001 – LogProducer

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

4.2.3.2 APP_TC_002 – LifeCycle::initialize InitializeError

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

4.2.3.3 APP_TC_003 – PortSupplier::GetPort() UnknownPort

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

4.2.3.4 APP_TC_004 – SCD describes Uses and Provides Ports

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

4.2.3.5 APP_TC_005 – SAD defines "external" interfaces

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

4.2.3.6 APP_TC_006 – ResourceFactory::releaseResource()

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

4.2.3.7 APP_TC_007 – ResourceFactory::shutdown()

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

4.2.3.8 APP_TC_008 – OS Services

This test case verifies AP0603. The objective of the test is to verify that the application is limited to using services that are designated mandatory in the SCA Application Environment Profile (AEP, SCA v2.2.2 Appendix B [Reference 4]). This test will identify AEP OS service violations in the source code.

4.2.3.9 APP_TC_009 – TestableObject::runTest() Properties

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

4.2.3.10 APP_TC_010 – TestableObject::runTest()

This test case verifies AP0079, AP0080, and AP0081. The objective of this test is to verify that the *runTest()* operation can be used to test component implementations. The *runTest ()* operation will use testID to determine the predefined test implementations and testValues for additional information to

the implementation-specific tests to be performed. This test will verify that the *runTest()* operation places a return value into the testValues parameter.

4.2.3.11 APP_TC_011 – PropertySet::configure()

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

4.2.3.12 APP_TC_012 – Port::disconnectPort()

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

4.2.3.13 APP_TC_013 – PortSupplier::getPort()

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

4.2.3.14 APP_TC_014 – Port::connectPort() InvalidPort

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

4.2.3.15 APP_TC_015 – Port::connectPort()

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

4.2.3.16 APP_TC_016 – TestableObject::runTest() Exceptions

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

4.2.3.17 APP_TC_017 – Resource::Identifier

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

4.2.3.18 APP_TC_018 – Domain Profile

This test verifies AP0588, AP0590, AP0591, AP0592, AP0593, and AP0594. The objective of this test is to verify that the Domain Profile is compliant to the Document Type Definitions (DTD) provided in Appendix D [Reference 6]. The test also verifies that the Domain Profile files have the

correct file extensions and that the first two lines of each Domain Profile file has the proper declaration.

If verifying SCA Extension requirements test case APP_TC_051 replaces this test case.

4.2.3.19 APP_TC_019 – LifeCycle::releaseObject()

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

4.2.3.20 APP_TC_020 – AEP, Ada Functionality

This test case verifies AP0655. The objective of this test is to verify that if the application is using the Ada language then it is restricted to using equivalent Ada functionality as defined in the ISO/IEC 14519:2001 POSIX Ada Language Interface specification. This can be done by implementing a C/C++ interface as described in Appendix B [Reference 4].

4.2.3.21 APP_TC_021 – High Order Languages

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

4.2.3.22 APP_TC_022 – AEP, Abnormal Termination

This test case verifies AP0661. Table B-6 of the SCA Appendix B [Reference 4], AEP specifies the mandatory POSIX Signals function to be provided by the OE. This test case should **only** be executed if the application waveform provides any of the signals functions in Table B-6 in which the functions over-ride the base POSIX functions provided by the OE. The SCA AEP specification requires that these signals functions when called must be handled properly so that other processes are not impacted. The objective of this test is to verify that the application waveform, which is the provider of these signals functions implements signal handlers for each signal function in order to properly manage process terminations in an environment that does not support multiple processes.

4.2.3.23 APP_TC_024 – Minimum CORBA

This test case verifies AP0607. The objective of this test is to verify that the application source code is limited to using CORBA as referenced in the Minimum CORBA specification of the SCA v2.2.2.

4.2.3.24 APP_TC_025 – PropertySet::query()

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

4.2.3.25 APP_TC_026 – Base Application Interfaces

This test case verifies AP0608. The objective of this test is to verify that each component identified as implementing the Base Application Interfaces (*Ports, LifeCycle, TestableObject, PropertySet, PortSupplier, Resource* or *ResourceFactory*) follow the specifications identified in section 3.1.3.1 of the SCA. Furthermore, this test verifies that the interfaces implemented match the IDL signature in Appendix C [Reference 5].

4.2.3.26 APP_TC_027 – ResourceFactory::createResource()

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

4.2.3.27 APP_TC_028 – Core Framework Interfaces

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

4.2.3.28 APP_TC_029 – Producer and Consumer of Events

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

4.2.3.29 APP_TC_030 – AEP, POSIX::kill()

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

4.2.3.30 APP_TC_031 – ResourceFactory::Identifier

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

4.2.3.31 APP_TC_032 – Resource::start() startError

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

4.2.3.32 APP_TC_033 – Resource::stop()

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

4.2.3.33 APP_TC_034 – Resource::stop() not permanent

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

4.2.3.34 APP_TC_035 – ResourceFactory::createResource() CreateResourceFailure

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

4.2.3.35 APP_TC_036 – Domain Profile files

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

4.2.3.36 APP_TC_037 – Standard arguments for executable components

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

4.2.3.37 APP_TC_039 – Requirements for IDL mappings

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

4.2.3.38 APP_TC_040 – General Rules (Legacy Software)

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

4.2.3.39 APP_TC_041 – OS Services (File access)

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

4.2.3.40 APP_TC_042 – OS Services (SIGQUIT)

This test case verifies AP0606. The objective of this test is to verify that all application processes shall have a handler registered for the POSIX-defined *SIGQUIT* signal. *SIGQUIT* is a controlled termination and is a necessary SCA functionality.

4.2.3.41 APP_TC_043 – Registering with the Naming Service

This test case verifies requirement AP0709. The objective of this test case is to verify that any module that is created during the execution of an application is registered successfully with the Naming Service.

4.2.3.42 APP_TC_044 – Usage of stringified IORs

This test case verifies AP0740. The objective of this test is to verify that the application does not use static stringified IOR's. Static stringifying of an IOR is accomplished by assigning (hard-coding) a static string value within the application – the value will always remain the same.

4.2.4 SCA v2.2.2 AEP Amended Manual Test Cases and Test Objectives – Summary

The planned tests that will verify the SCA v2.2.2A AEP Amended requirements compliance are listed in the following sections. Requirements are logically grouped together into test cases, when appropriate. Each test case name follows the naming convention described in section 4.2.1

The following subsections list the SCA AEP Amended application test cases, the functional area(s), and objectives.

4.2.4.1 APP_TC_047 - AEP Amended Ada functionality

This test case verifies AP0792. The objective of this test is to verify that if the application is using the Ada language then it is restricted to using equivalent Ada functionality as defined in the ISO/IEC 14519:2001 POSIX Ada Language Interface specification. Annex C of the IEEE document shows the Ada to C language cross references.

Note: This test case replaces APP_TC_020 when the testing event is for SCA AEP Amended verification.

4.2.4.2 APP_TC_048 - AEP Amended Applications have no abnormal termination

This test case verifies AP0798. Table B-6 of the SCA Appendix B [Reference 4], AEP Amended specifies the mandatory POSIX Signals function to be provided by the OE. This test case should only be executed if the application waveform provides any of the signals functions in Table B-6 in which the functions over-ride the base POSIX functions provided by the OE. The SCA AEP Amended specification requires that these signals functions when called must be handled properly so that other processes are not impacted. The objective of this test is to verify that the application waveform, which is the provider of these signals functions implements signal handlers for each signal function in order to properly manage process terminations in an environment that does not support multiple processes.

Note: This test case replaces APP_TC_022 when the testing event is for SCA AEP Amended verification.

4.2.4.3 APP_TC_049 - AEP Amended kill function limitations

This test case verifies AP0799. The objective of this test is to verify that the kill() function is not called with a negative argument. The kill() function uses two arguments, namely, pid and signal, to send a signal to a process or a group of processes specified by the pid (process ID). The second argument, signal, cannot be negative.

Note: This test case replaces APP_TC_030 when the testing event is for SCA AEP Amended verification.

4.2.4.4 APP_TC_050 - AEP Amended mandatory functions

This test case verifies AP0603. The objective of the test is to verify that the application is limited to using services that are designated mandatory in the SCA v2.2.2 Application Environment Profile (AEP), version v2.2.2A, <ICWG Approved>.

Note: This test case replaces APP_TC_008 when the testing event is for SCA AEP Amended verification.

4.2.5 SCA v2.2.2 Extensions Manual Test Cases and Test Objectives – Summary

The planned tests that will verify the SCA v2.2.2 Extension Requirements compliance are listed in the following sections. Requirements are logically grouped together into test cases, when appropriate. Each test case name follows the naming convention described in section 4.2.1

The following subsection lists the SCA Extension application test case, the functional area, and the objective.

4.2.5.1 APP_TC_051 – Domain Profile for SCA Extensions

This test verifies AP0588, AP0590, AP0591, AP0592, AP0593, AP0594, and AP0789. The objective of this test is to verify that the Domain Profile is compliant to the Document Type Definitions (DTD) provided in SCA Appendix D [Reference 6] and SCA Extensions document. Specifically the SAD and SCD files have been updated for the SCA Extension requirements. The test also verifies that the Domain Profile files have the correct file extensions and that the first two lines of each Domain Profile file has the proper declaration.

Note: This test case replaces APP_TC_018 when the testing event is for SCA Extension verification.

4.2.6 Test Cases -- Details

The test description provides 'what' is being tested – not the 'how'. The detailed test descriptions are contained in Appendix B. Included are the test case name, test objective and the name of the associated automated test name (if relevant). The step-by-step manual test procedures are also listed.

Generally, the procedures in Appendix B will use the test case's test description as the 'headers' and will then be followed by the detailed steps. Each row of the steps will contain what the tester needs to do (inspect, verify, locate, etc.). Additional columns contain the expected results of the step and the actual test results. The comment column contains any additional test information for clarification.

The last column, 'Test Result', allows the tester to describe the results of the test. Possible test results are '(P)ass, (F)ail, (U)ntested, or N/A'.

Each test case contains a Test Recording log to aid the tester in performing a test and to provide test artifacts. This Log Sheet contains an expandable table to record the test information associated with the relevant procedures. It is especially useful when there are an unknown number of items being looked at. For example, if a procedure needs to 'find all resources', then the results can be logged into multiple tables.

5 Requirements Traceability

This section provides the traceability between each test case and to the SCA v2.2.2 requirements being tested. The table with the traceability matrix, Table 5, may list test cases multiple times since the matrix is ordered by test case and there may be multiple requirements being tested in a single test case. The following describes the type of information provided in the traceability matrix.

The SCA v2.2.2 requirements contain identifiers which categorize the requirement as being applicable to the Application, the Operating Environment or to both. When the requirement is applicable to both the Application and Operating Environment, there will be two requirement identifiers with the same 4-digit number, one prefaced with "AP" and one prefaced with "OE".

Some SCA v2.2.2 requirements are associated with one or more criterion/criteria which is defined by SCA v2.2.2 as "is" or "are" statements and which indicate required condition(s). In SCA v2.2 criterion/criteria are described as behaviors. JTEL considers criteria statements as testable requirements. It is expected that both the requirement and its associated criterion/criteria concurrently pass together. Each criterion is identified by the identifier associated with the requirement followed by the "C" character and two digits.

Table 3 below illustrates the different tags with their meanings.

Table 3: Requirement Tagging Convention

SCA Requirements Tagging System		
Tag	**Meaning**	**Example**
AP	Applications requirements	AP0010
OE	Operating Environment requirements	OE0001
AP####C##	Criterion associated with an application requirement	AP0010C01 or OE0001C01

5.1 Test Case Mapping

This traceability section further describes the test case mapping from the previous Section 4 by identifying the test case, the identifier tag of the primary SCA requirement, the requirement text as well as the requirement type and test method. The table is sorted first with the test case name (column 1) and then by its tag (column 2).

Table 4 describes each of the columns in the Requirements Traceability test case matrix of Table 5.

Table 4: Explanation of Columns of Traceability Matrix

Column	Contains
SCA Tag	Identifier for the SCA v2.2.2 requirement which denotes the requirement relating to the application or to the OE or an associated criteria. The tag is appended with a number unique to the SCA v2.2.2 requirement.
Criteria Tag	Indication that there is a criteria associated with the requirement
Test Case Number	Test Case Number, comprised of APP(LICATION)_T(est)C(ase)_xxx, where number is the test case number
Requirement Text	Text of the requirement. References to appendices refer to the appendices of the SCA v2.2.2 specification
Section Number	SCA v2.2.2 Specification section number

Table 5: Requirement Traceability Table

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0011		APP_TC_001	A log producer shall only output log records that contain an enabled CosLwLog::LogLevel value.	3.1.2.2.1
AP0012		APP_TC_001	Log producers shall use their component identifier attribute in the producerId field of the CosLwLog:: ProducerLogRecord.	3.1.2.2.1
AP0013		APP_TC_001	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.	3.1.2.2.1
AP0063		APP_TC_029	A component (e.g., Resource, DomainManager, etc.) that consumes events shall implement the CosEventComm PushConsumer interface.	3.1.2.3.1
AP0064		APP_TC_029	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.	3.1.2.3.1
AP0065		APP_TC_029	A producer component shall not forward or raise any exceptions when the connection to a CosEventComm PushConsumer is a nil or invalid reference.	3.1.2.3.1
AP0069		APP_TC_015	The connectPort operation shall make a connection to the component identified by its input parameters.	3.1.3.1.1.5.1.3

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0070		APP_TC_014	The connectPort operation shall raise the InvalidPort exception when the input connection parameter is an invalid connection for this port.	3.1.3.1.1.5.1.5
AP0070	C004	APP_TC_014	The InvalidPort exception indicates one of the following errors has occurred in the specification of a Port association: 1. errorCode 1 means the Port component is invalid (unable to narrow object reference) or illegal object reference.	3.1.3.1.1.3.1
AP0071		APP_TC_015	The connectPort operation shall raise the OccupiedPort exception when unable to accept the connections because the port is already fully occupied.	3.1.3.1.1.5.1.5
AP0072		APP_TC_012	The disconnectPort operation shall break the connection to the component identified by the input connectionId parameter.	3.1.3.1.1.5.2.3
AP0073		APP_TC_012	The disconnectPort operation shall raise the InvalidPort exception when the input connectionId parameter is not a known connection to the Port component.	3.1.3.1.1.5.2.5
AP0073	C003	APP_TC_012	The InvalidPort exception indicates one of the following errors has occurred in the specification of a Port association: 2. errorCode 2 means the Port name is not found (not used by this Port).	3.1.3.1.1.3.1
AP0074		APP_TC_002	The initialize operation shall raise an InitializeError exception when an initialization error occurs.	3.1.3.1.2.5.1.5
AP0075		APP_TC_019	The releaseObject operation shall release all internal memory allocated by the component during the life of the component.	3.1.3.1.2.5.2.3
AP0076		APP_TC_019	The releaseObject operation shall tear down the component and release it from the CORBA environment.	3.1.3.1.2.5.2.3
AP0078		APP_TC_019	The releaseObject operation shall raise a ReleaseError exception when a release error occurs.	3.1.3.1.2.5.2.5
AP0079		APP_TC_010	The runTest operation shall use the input testId parameter to determine which of its predefined test implementations should be performed.	3.1.3.1.3.5.1.3
AP0080		APP_TC_010	The id/value pair(s) of the testValues parameter shall be used to provide additional information to the implementation-specific test to be run.	3.1.3.1.3.5.1.3
AP0081		APP_TC_010	The runTest operation shall return the result(s) of the test in the testValues parameter.	3.1.3.1.3.5.1.3

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0082		APP_TC_009	Valid testId(s) and both input and output testValues (properties) for the runTest operation shall at a minimum be the test properties defined in the properties test element of the component's Properties Descriptor (refer to SCA Appendix D [Reference 6] Domain Profile).	3.1.3.1.3.5.1.3
AP0083		APP_TC_009	All testValues parameter properties (i.e., test properties defined in the propertyfile(s) referenced in the component's SPD) shall be validated.	3.1.3.1.3.5.1.3
AP0084		APP_TC_016	The runTest operation shall not execute any testing when the input testId or any of the input testValues are not known by the component or are out of range.	3.1.3.1.3.5.1.3
AP0085		APP_TC_016	The runTest operation shall raise the UnknownTest exception when there is no underlying test implementation that is associated with the input testId given.	3.1.3.1.3.5.1.5
AP0086		APP_TC_016	The runTest operation shall raise the CF UnknownProperties exception when the input parameter testValues contains any CF DataTypes that are not known by the component's test implementation or any values that are out of range for the requested test.	3.1.3.1.3.5.1.5
AP0087		APP_TC_016	The exception parameter invalidProperties shall contain the invalid testValues properties id(s) that are not known by the component or the value(s) are out of range.	3.1.3.1.3.5.1.5
AP0089		APP_TC_013	The getPort operation shall return the CORBA object reference that is associated with the input port name.	3.1.3.1.4.5.1.4
AP0090		APP_TC_003	The getPort operation shall raise an UnknownPort exception if the port name is invalid.	3.1.3.1.4.5.1.5
AP0091		APP_TC_011	The configure operation shall assign values to the properties as indicated in the input configProperties parameter.	3.1.3.1.5.5.1.3
AP0092		APP_TC_011	Valid properties for the configure operation shall at a minimum be the configure readwrite and writeonly properties referenced in the component's SPD.	3.1.3.1.5.5.1.3
AP0093		APP_TC_011	The configure operation shall raise a PartialConfiguration exception when some configuration properties were successfully set and some configuration properties were not successfully set.	3.1.3.1.5.5.1.5

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0094		APP_TC_011	The configure operation shall raise an InvalidConfiguration exception when a configuration error occurs and no configuration properties were successfully set.	3.1.3.1.5.5.1.5
AP0095		APP_TC_025	The query operation shall return all component properties when the inout parameter configProperties is zero size.	3.1.3.1.5.5.2.3
AP0096		APP_TC_025	The query operation shall return only those id/value pairs specified in the configProperties parameter if the parameter is not zero size.	3.1.3.1.5.5.2.3
AP0097		APP_TC_025	Valid properties for the query operation shall be all configure properties (simple properties whose kind element's kindtype attribute is "configure") whose mode attribute is "readwrite" or "readonly" and any allocation properties with an action value of "external" as referenced in the component's SPD.	3.1.3.1.5.5.2.3
AP0098		APP_TC_025	The query operation shall raise the CF UnknownProperties exception when one or more properties being requested are not known by the component.	3.1.3.1.5.5.2.5
AP0099		APP_TC_032	The errorNumber parameter shall indicate a CF ErrorNumberType value.	3.1.3.1.6.3.1
AP0100		APP_TC_033	The errorNumber parameter shall indicate a CF ErrorNumberType value.	3.1.3.1.6.3.2
AP0101		APP_TC_017	The readonly identifier attribute shall contain the unique identifier for a Resource instance.	3.1.3.1.6.4.1
AP0102		APP_TC_033	The stop operation shall disable all current operations and put the resource in a non-operating condition.	3.1.3.1.6.5.2.3
AP0103		APP_TC_033	The stop operation shall raise the StopError exception if an error occurs while stopping the resource.	3.1.3.1.6.5.2.5
AP0105		APP_TC_032	The start operation shall raise the StartError exception if an error occurs while starting the resource.	3.1.3.1.6.5.1.5
AP0107		APP_TC_007	The shutdown operation shall raise the ShutdownFailure exception when processing errors prevent the release of the resource factory from the CORBA environment or when all resources have not been released from the resource factory.	3.1.3.1.7.5.3.5
AP0108		APP_TC_035	The error number shall indicate a CF ErrorNumberType value.	3.1.3.1.7.3.3
AP0111		APP_TC_027	The createResource operation shall return a reference to the created resource.	3.1.3.1.7.5.1.4

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0115		APP_TC_035	The createResource operation shall raise the CreateResourceFailure exception when it cannot create the resource.	3.1.3.1.7.5.1.5
AP0116		APP_TC_006	The releaseResource operation shall decrement the reference count for the specified resource, as indicated by the resourceId parameter.	3.1.3.1.7.5.2.3
AP0117		APP_TC_006	The releaseResource operation shall release the resource from the CORBA environment and make the resource no longer available when the resource's reference count is zero.	3.1.3.1.7.5.2.3
AP0118		APP_TC_006	The releaseResource operation shall raise the InvalidResourceId exception if an invalid resourceId is received.	3.1.3.1.7.5.2.5
AP0119		APP_TC_007	The shutdown operation shall release the resource factory from the CORBA environment and make it unavailable to any subsequent calls to its object reference.	3.1.3.1.7.5.3.3
AP0588		APP_TC_018	Domain Profile files shall be complaint to the Document Type Definitions (DTDs) provided in Appendix D [Reference 6].	3.1.3.5
AP0588		APP_TC_051	Domain Profile files shall be complaint to the Document Type Definitions (DTDs) provided in Appendix D[Reference 6]. (SCA Extension verification)	3.1.3.5
AP0590		APP_TC_018	All XML files shall have as the first two lines as an XML declaration (?xml) and a document type declaration (!DOCTYPE).	3.1.3.5
AP0590		APP_TC_051	All XML files shall have as the first two lines as an XML declaration (?xml) and a document type declaration (!DOCTYPE). (SCA Extension verification)	3.1.3.5
AP0591		APP_TC_018	A Software Package Descriptor file shall have a ".spd.xml" extension.	3.1.3.5.1
AP0591		APP_TC_051	A Software Package Descriptor file shall have a ".spd.xml" extension. (SCA Extension verification)	3.1.3.5.1
AP0592		APP_TC_018	A Software Component Descriptor file shall have a ".scd.xml" extension.	3.1.3.5.2
AP0592		APP_TC_051	A Software Component Descriptor file shall have a ".scd.xml" extension. (SCA Extension verification)	3.1.3.5.2
AP0593		APP_TC_018	A Software Assembly Descriptor file shall have a ".sad.xml" extension.	3.1.3.5.3
AP0593		APP_TC_051	A Software Assembly Descriptor file shall have a ".sad.xml" extension. (SCA Extension verification)	3.1.3.5.3

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0594		APP_TC_018	A Properties File shall have a ".prf.xml" extension.	3.1.3.5.4
AP0594		APP_TC_051	A Properties File shall have a ".prf.xml" extension. (SCA Extension verification)	3.1.3.5.4
AP0603		APP_TC_008	Applications shall be limited to using the OS services that are designated as mandatory in the SCA Application Environment Profile (Appendix B [Reference 4]).	3.2.1.1
AP0603		APP_TC_050	Applications shall be limited to using the OS services that are designated as mandatory in the SCA Application Environment Profile (SCA Appendix B [Reference 4], Amended verification).	B.4.1.15
AP0604		APP_TC_041	Applications shall perform file access through the CF File interfaces.	3.2.1.1
AP0606		APP_TC_042	All application processes shall have a handler registered for the POSIX-defined SIGQUIT signal.	3.2.1.1
AP0607		APP_TC_024	Applications shall be limited to using CORBA and CORBA services defined in the referenced minimumCORBA specification.	3.2.1.2
AP0608		APP_TC_026	Applications shall implement the Base Application Interfaces as specified in section 3.1.3.1 using the corresponding IDL in Appendix C [Reference 5].	3.2.1.3
AP0609		APP_TC_028	Each application component shall support the mandatory Naming Context IOR, Name Binding, and the identifier execute parameters as described in 3.1.3.2.2.5.1, in addition to their user-defined execute properties in the component's SPD.	3.2.1.3
AP0610		APP_TC_028	Each application component shall bind its object reference to the Naming Context IOR using the Name Binding parameter.	3.2.1.3
AP0611		APP_TC_028	Each executable component of an application shall set its identifier attribute using the component identifier execute parameter.	3.2.1.3
AP0612		APP_TC_037	Each executable component of an application shall accept the standard argv arguments of the POSIX exec family of functions .	3.2.1.3
AP0613		APP_TC_036	An application, each application component, and each device manager shall be accompanied by the appropriate Domain Profile files per section 3.1.3.5.	3.2.1.3
AP0613	C145	APP_TC_036	The software profile for an application consists of one SAD file that references (directly or	D.1

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
			indirectly) one or more SPD, SCD, and properties (PRF) files.	
AP0613	C148	APP_TC_036	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.	D.2.1
AP0613	C149	APP_TC_036	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.	D.2.1
AP0613	C150	APP_TC_036	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.	D.2.1
AP0613	C151	APP_TC_036	Duplicated properties must be the same property type, only the value can be over-ridden.	D.2.1
AP0613	C152	APP_TC_036	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.	D.2.1
AP0613	C153	APP_TC_036	The propertyfile element is used to indicate the local filename of the Property Descriptor file associated with the Software Package.	D.2.1.4
AP0613	C154	APP_TC_036	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.	D.2.1.4.1
AP0613	C155	APP_TC_036	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).	D.2.1.5

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0613	C156	APP_TC_036	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.	D.2.1.6
AP0613	C157	APP_TC_036	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.	D.2.1.6
AP0613	C158	APP_TC_036	The stack size and priority are options parameters used by the CF ExecutableDevice execute() operation.	D.2.1.6.3
AP0613	C159	APP_TC_036	Data types for the values of these options are unsigned long.	D.2.1.6.3
AP0613	C160	APP_TC_036	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."	D.2.1.6.3
AP0613	C161	APP_TC_036	The softpkgref element 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.	D.2.1.6.9.1
AP0613	C162	APP_TC_036	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.	D.2.1.6.9.2
AP0613	C163	APP_TC_036	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.	D.2.1.7
AP0613	C164	APP_TC_036	The id attribute for a simple property that is an allocation type is a DCE UUID value, as specified in section D.2.1	D.4.1.1
AP0613	C165	APP_TC_036	Only simple elements can be used as execparam types.	D.4.1.1.6
AP0613	C166	APP_TC_036	At a minimum, the component interface has to be a CF Resource, CF ResourceFactory, or CF Device interface.	D.5.1.4

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0613	C167	APP_TC_036	The softwareassembly element's id attribute is a DCE UUID, as specified in section D.2.1, which uniquely identifies the assembly.	D.6.1
AP0613	C168	APP_TC_036	The componentfiles element is used to indicate that an assembly is made up of 1…n component files. The componentfile element contains a reference to a local file, which is a Software Package Descriptor file.	D.6.1.2
AP0613	C169	APP_TC_036	The componentfileref element is used to reference a particular Software PackageDescriptor file. The componentfileref element's refid attribute corresponds to the componentfile element's id attribute.	D.6.1.3.2
AP0613	C170	APP_TC_036	The componentinstantiation's id attribute is a DCE UUID that uniquely identifies the component. The id is a DCE UUID value as specified in section D.2.1.	D.6.1.3.3
AP0613	C171	APP_TC_036	The optional findcomponent element should be specified except when there is no CORBA object reference for the component instance (e.g., DSP code).	D.6.1.3.3
AP0613	C172	APP_TC_036	The optional findcomponent element is used to obtain the CORBA object reference for the component instance. The two sources for obtaining a CORBA object reference are: 1. The componentresourcefactoryref element, which refers to a particular CF ResourceFactory componentinstantiation element found in the SAD, which is used to obtain a CF Resource instance for this componentinstantiation element. The refid attribute refers to a unique componentinstantiation id attribute. The componentresourcefactoryref element contains an optional resourcefactoryproperties element, which specifies the properties "qualifiers", for the CF ResourceFactory create call." 2. The CORBA Naming Service, which is used to find the component's CORBA object reference. The name specified in the namingservice element is a partial name that is used by the CF ApplicationFactory to form the complete context name.	D.6.1.3.3
AP0613	C173	APP_TC_036	The usesidentifier element identifies which "uses port" on the component is to participate in the connection relationship. This identifier will correspond with an id for one of the component	D.6.1.5.1.1.1

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
			ports specified in the Software Component Descriptor.	
AP0613	C174	APP_TC_036	The devicethatloadedthis-componentref element refers to a specific component found in the assembly, which is used to obtain the logical CF Device that was used to load the referenced component from the CF ApplicationFactory.	D.6.1.5.1.1.6
AP0613	C175	APP_TC_036	The findby element by itself is used when the object reference is not a CF Resource type.	D.6.1.5.1.2
AP0613	C176	APP_TC_036	The supportedidentifier element identifies which supported interface on the component is to participate in the connection relationship. This identifier will correspond with the repid attribute of one of the component's supportsinterface elements, specified in the Software Component Descriptor.	D.6.1.5.1.3.1
AP0614		APP_TC_004	Interfaces provided by a component shall be described in a Software Component Descriptor file as provides ports.	3.2.2
AP0615		APP_TC_004	Interfaces required by a component shall be described in a Software Component Descriptor file as uses ports.	3.2.2
AP0616		APP_TC_005	An application interface shall be referenced in the application's SAD externalports element, and thus declared "external", if the interface provides a service that is used by other applications.	3.2.2
AP0628		APP_TC_021	Software developed for an SCA-compliant system shall be developed in a standard higher order language.	3.4
AP0630		APP_TC_040	Legacy software shall interface with the Core Framework in accordance with this specification.	3.4
AP0655		APP_TC_020	Any Ada application shall be restricted to using the equivalent Ada functionality, as defined in POSIX Ada language binding (ISO/IEC) 14519:2001), designated as mandatory by the AEP or may use the C interface	B.4
AP0661		APP_TC_022	An application that conforms to the AEP shall not result in abnormal termination of the process because this profile does not support multiple processes.	B.4.1.4
AP0662		APP_TC_030	An application that conforms to the AEP shall not call the kill( ) function with a negative argument because this profile does not require process group functionality.	B.4.1.4
AP0704		APP_TC_034	The stop operation shall not inhibit subsequent configure, query, and start operations.	3.1.3.1.6.5.2.3

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0705		APP_TC_031	The readonly identifier attribute shall contain the unique identifier for a ResourceFactory instance.	3.1.3.1.7.4.1
AP0706		APP_TC_027	If the resource already exists, the createResource operation shall return a reference to the existing resource.	3.1.3.1.7.5.1.4
AP0709		APP_TC_043	Upon execution of a software module by the create operation, a Resource or a ResourceFactory component shall register with the Naming Service.	3.1.3.2.2.5.1.3
AP0740		APP_TC_044	Applications shall not utilize static stringified IORs.	3.2.1.2
AP0741		APP_TC_039	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.	3.2.2
AP0742		APP_TC_039	All SCA APIs shall have their interfaces described in IDL.	3.2.2.1
AP0743		APP_TC_039	All non-IDL interfaces shall provide an IDL mapping within the service definition.	3.2.2.1
AP0747		APP_TC_001	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.	3.1.2.2.1
AP0748		APP_TC_027	The createResource operation shall create a resource if no resource exists for the given resourceId and shall assign the given resourceId to a new resource.	3.1.3.1.7.5.1.3
AP0753		APP_TC_027	The createResource operation shall set a reference count to one, when the resource is initially created, or increment the reference count by one, when the resource already exists.	3.1.3.1.7.5.1.3
AP0789		APP_TC_051	An Application Deployment Descriptor file shall have an ".add.xml" extension.	SCA Extensions, section 2.9
AP0792		APP_TC_047	Any Ada application shall be restricted to using the equivalent Ada functionality, as defined in POSIX Ada language binding (ISO/IEC 14519:2001), designated as mandatory by the AEP or may use the C interface.	SCA Appendix B [Reference 4] Amended, B.4
AP0798		APP_TC_048	An application that conforms to the AEP shall not result in abnormal termination of the process because this profile does not support multiple processes.	SCA Appendix B[Reference 4] Amended, B.4.1.4

SCA Tag	Criteria Tag	Test Case Number	Requirement Text	Section Number
AP0799		APP_TC_050	An application that conforms to the AEP shall not call the kill() function with a negative argument because this profile does not require process group functionality.	SCA Appendix B[Reference 4] Amended, B.4.1.4

Appendix A: Acronym List

The following are the acronyms that are used in this document:

Acronym	Definition
AASTD	Automated Applications Software Test Description
ADD	Application Deployment Descriptor
AEP	Application Environment Profile
AOESTD	Automated Operating Environment Software Test Description
API	Application Program Interface
CF	Core Framework
CORBA	Common Object Request Broker Architecture
DISA	Defense Information Systems Agency
DSP	Digital Signal Processor
DTD	Document Type Definitions
GUTS	General Unit Test Suite
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
ICD	Interface Control Document
IOR	Interoperable Object Reference
ISO/IEC	International Organization for Standardization / International Engineering Consortium
JITC	Joint Interoperability Test Command
JTAP	JTNC Test Application
JTEL	JTNC Test and Evaluation Laboratory
JTNC	Joint Tactical Networking Center

Acronym	Definition
JTR	Joint Tactical Radio
MASTD	Manual Application Software Test Description
OE	Operating Environment
OMG	Object Management Group
ORD	Operational Requirements Document
OS	Operating System
POSIX	Portable Operating System Interface
PRF	Properties File
SAD	Software Assembly Descriptor
SCA	Software Communications Architecture
SCD	Software Component Descriptor
SDD	Software Design Document
SRD	Support and Rationale Document for the SCA
SRS	Software Requirements Specification
SSC SD	Space and Naval Warfare Systems Center San Diego
STD	Software Test Description
SUM	Software User Manual
SVD	Software Version Description
T&E	Test and Evaluation
TA	Test Authority
XML	eXtensible Markup Language

Appendix B: Test Procedures

[Appendix B is provided as a separate document.]

Appendix C: AEP Amended Requirements Test Procedures

[Appendix C is provided as a separate document.]

Appendix D: SCA Extension Requirements Test Procedures

[Appendix D is provided as a separate document.]