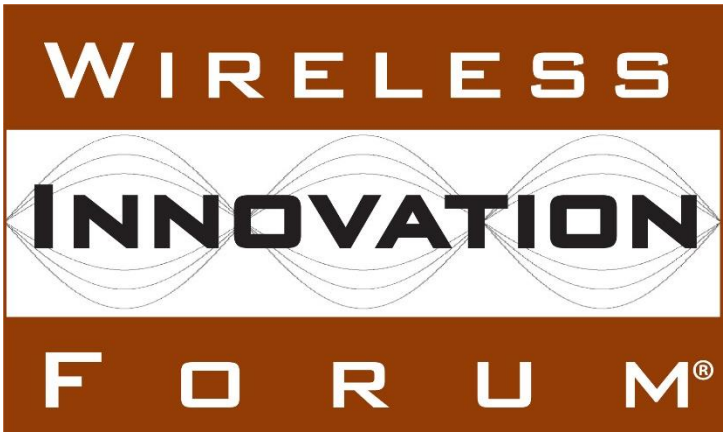

SCA 4.1 Test Procedures

Document WINNF-TS-4001

Version V1.0.0
18 December 2018

TERMS, CONDITIONS & NOTICES

This document has been prepared by the Work Group of WInnF Project SCA-2017-001 "Verification of SCA 4.1 Applications" to assist The Software Defined Radio Forum Inc. (or its successors or assigns, hereafter "the Forum"). It may be amended or withdrawn at a later time and it is not binding on any member of the Forum or of the SCA Test and Evaluation Work Group.

Contributors to this document that have submitted copyrighted materials (the Submission) to the Forum for use in this document retain copyright ownership of their original work, while at the same time granting the Forum a non-exclusive, irrevocable, worldwide, perpetual, royalty-free license under the Submitter's copyrights in the Submission to reproduce, distribute, publish, display, perform, and create derivative works of the Submission based on that original work for the purpose of developing this document under the Forum's own copyright.

Permission is granted to the Forum's participants to copy any portion of this document for legitimate purposes of the Forum. Copying for monetary gain or for other non-Forum related purposes is prohibited.

THIS DOCUMENT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS DOCUMENT.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

This document was developed following the Forum's policy on restricted or controlled information (Policy 009) to ensure that that the document can be shared openly with other member organizations around the world. Additional Information on this policy can be found here: http://www.wirelessinnovation.org/page/Policies_and_Procedures

Although this document contains no restricted or controlled information, the specific implementation of concepts contain herein may be controlled under the laws of the country of origin for that implementation. Readers are encouraged, therefore, to consult with a cognizant authority prior to any further development.

Wireless Innovation Forum ™ and SDR Forum ™ are trademarks of the Software Defined Radio Forum Inc.

Table of Contents

List of Figures

List of Tables

Contributors

The following individuals and their organization of affiliation are credited as Contributors to development of the specification, for having been involved in the work group that developed the draft then approved by WInnF member organizations:

- Ed Brabant, JTEL
- Huan Dao, JTEL
- James Evangelos, JTNC Standards
- Chris Hagen, Rockwell Collins
- Olivier Kirsch, Kereval
- Frédéric Le Roy, ENSTA Bretagne
- François Lévesque, NordiaSoft
- Charles Linn, Harris
- Eric Nicollet, Thales Communication & Security
- Kevin Richardson, JTNC Standards
- Sarvpreet Singh, Fraunhofer FKIE
- Jonathan Springer, Reservoir Labs
- James Ezick, Reservoir Labs

SCA 4.1 Test Procedures

1 Introduction

This document contains test procedures that may be used to verify that application implementations are compliant with the SCA 4.1 specification [Ref0]. The SCA requirements that are verified by these procedures are those listed in Table 1 and Table 2 of the Application Verification Plan (AVP) document [Ref1]. Verification approaches were selected for each requirement to satisfy the objective of minimizing the duration and expense of SCA 4.1 verification as described in the AVP. Therefore, the preferred methods of verification, unless otherwise specified, are Test and Analysis since they allow for full or partial automation. If neither of those methods can be used, Demonstration is preferred over the Inspection. The Inspection method should only be used when no other automated or semi-automated approach can be applied to verify a requirement under test.

The remainder of the document is structured as follows. Section 2 presents common rules to be applied during the execution of the test procedures, along with supplemental terms and definitions. Sections 3 and 4 contain contextual information and preconditions that are used across multiple test procedures. Section 5 contains the SCA application requirements and their associated test procedures. Lastly, section 6 contains generic sub-procedures that are utilized within many of the test procedures.

The following terms are used within this document and should be interpreted as described in RFC-2119:

- SHALL is a mandatory requirement (negative is SHALL NOT)
- SHOULD is recommended requirement/best practice (negative is SHOULD NOT)
- MAY is an optional requirement, i.e., something that is allowed (negative is NEED NOT)

2 Verification Definitions and Rules

2.1 Terms and definitions

Abstract Components: SCA 4.1 [Ref0] abstract component (i.e. BaseComponent, BaseFactoryComponent) requirements are written as if they are implemented, packaged and deployed by the abstract component. For example, requirement SCA430 is written as follows:

> *"A BaseComponent shall supply ports for all the ports defined in its domain profile."*

Within the SCA 4.1 Test Procedures, such requirements are allocated to the **component** that inherits the abstract component, thus the one that implements (realizes) the inherited interfaces. Given that an SCA 4.1 ManageableApplicationComponent inherits from BaseComponent, SCA430 can be interpreted in the following manner:

> *"A ManageableApplicationComponent shall supply ports for all the ports defined in its domain profile."*

This interpretation is contingent upon the deployed ManageableApplicationComponent implementing the Connectable Unit of Functionality (UoF). The particulars of the implementation (e.g. does the component inherit that CF::PortAccessor interface directly) are implementation dependent and not prescribed by the test procedures.

Component: When a test procedure references a "component", it is a concrete application component, i.e. an application component defined in SCA 4.1 [Ref0] section 3.1.3.2.2, "Components".

Instantiation Reference: A reference to an instance of a component or nested application defined within the SAD. An instantiation reference is designated by a componentinstantiationref or assemblyinstantiationref element and points to the definition of a component instance or a nested application within the application.

SAD File: The SAD file is the software profile of an application. A SAD may contain references to other SAD files when nested applications are used (thus forming a hierarchy of applications). The root SAD is defined as the descriptor file referenced by the profileFileName input parameter of the CF::DomainInstallation::installApplication() operation. The root SAD contains an assemblyplacement element for each instance of a nested application, if nested applications are implemented. A nested SAD may also contain nested applications.

If the application under test contains nested application(s):

- When a test procedure references a SAD file without explicitly specifying "root SAD", the term applies to any SAD file referenced by the application.
- When a test procedure refers to the SAD file of an application component, the SAD file is the one which references the application component's SPD file.

Class(es) used to implement a component: A component is implemented through one or more source files using one or more classes. When a test procedure analyzes or inspects source code to validate a requirement, the term "for each class" means that the procedure will consider all source files (and classes) provided, no matter where they are used in the implementation of the component.

2.2 Common rules for the execution of the test procedures

1. A test procedure is applicable to a product if the product is an applicable SCA component for the requirement (as listed in Table 1 or 2 of the AVP document) and

    a. the associated requirement is listed in Table 1 of the AVP document

    b. or the associated requirement is listed in Table 2 of the AVP document and the component claims to be conform to the applicable UoF for the requirement.

2. Each test procedure is an independent entity that provides any contextual information, facts, assumptions, dependencies, or interpretations which are necessary to understand or execute the procedure.

3. A test procedure may have preconditions, which identify conditions that must be satisfied before a procedure can be executed. When any of a procedure's preconditions cannot be satisfied, the procedure should not be executed and the result of the test procedure will be considered "unverified" (i.e. applicable but unverified).

4. A test procedure may be written to verify more than one requirement. In those instances, each step identified within the procedure must be performed as part of verification unless a requirement number, specified within parenthesis, is provided at the end of the step action. A verification step containing a requirement number indicates that the step should only be executed when the test procedure is used to verify that requirement. When the procedure is used to verify a different requirement, such a step must be omitted.

5. Unless otherwise stated, the steps within a test procedure are executed sequentially. After a test step is successful, the execution continues with the next step, until all test steps are performed or as directed. If a test step fails, the verification result for the requirement under test will be a failure, and no further test steps need be performed. If all executed test steps are successful, then the verification result of the requirement under test procedure will be a pass.

3 Common Context Groups

3.1 Common Context for SCA 4.1 Test Platform

The following assertions apply to the "Context" section of all test procedures involving runtime execution of an application and/or its components.

1. The SCA 4.1 Test Platform:
   1.1. conforms to the SCA 4.1 specification with all UoFs an OE may support, i.e. equivalent functionality to the SCA 4.1 "Full" OE profile, when interfacing with the application and/or its components.
   1.2. implements the SCA 4.1 "Full" CORBA/e and RT CORBA profiles
   1.3. implements the SCA 4.1 "Full" AEP profile
   1.4. supports loading and executing the application under test

3.2 Common Context related to the validation of an application component

The following assertions apply to the "Context" section of all test procedures for requirements associated with an application component.

1. This test procedure will start with the root SAD of the application.

3.3 Common Context for requirements associated with BaseComponent (including a sub-application)

The following assertions apply to the "Context" section of all test procedures for requirements associated with BaseComponent.

1. The BaseComponent is a component part of an application under test.
2. A component is identified by the <componentplacement> or <assemblyplacement> element in the SAD of the application under test.
3. The domain profile for the component is the file referenced by an element stated in item 2 and the SPD, SCD and PRF(s) it references.

3.4 Common Context for requirements associated with BaseComponent (not including a sub-application)

The following assertions apply to the "Context" section of all test procedures for requirements associated with BaseComponent.

1. The BaseComponent is a component part of an application under test.
2. A component is identified by the <componentplacement> element in the SAD of the application under test.
3. The domain profile for the component is the file referenced by an element stated in item 2 and the SPD, SCD and PRF(s) it references.

3.5 Common Context for requirements associated with ManageableApplicationComponent

The following assertions apply to the "Context" section of all test procedures for requirements associated with ManageableApplicationComponent.

1. The ManageableApplicationComponent is a component part of an application under test.
2. A ManageableApplicationComponent is identified by the <componentplacement> element in the SAD of the application under test.
3. The SCD file for the component contains a <componenttype> element value of MANAGEABLE_APPLICATION_COMPONENT.
4. The domain profile for the ManageableApplicationComponent is the file referenced by an element stated in item 2 and the SPD, SCD and PRF(s) it references.

3.6 Common Context for requirements associated with ApplicationControllerComponent

The following assertions apply to the "Context" section of all test procedures for requirements associated with ApplicationControllerComponent.

1. The ApplicationControllerComponent is a component part of an application under test.
2. An application controller component is identified by the <assemblycontroller> element in the SAD of the application under test.
3. The <assemblycontroller> element stated in item 2 references a <componentplacement> in the SAD.
4. The domain profile for the ApplicationControllerComponent is the file referenced by an element stated in item 3 and the SPD, SCD and PRF(s) it references.

3.7 Common Context for requirements associated with BaseFactoryComponent

The following assertions apply to the "Context" section of all test procedures for requirements associated with BaseFactoryComponent.

1. The BaseFactoryComponent is a component part of an application under test.
2. A component is identified by the <componentplacement> element in the SAD of the application under test.
3. The SCD file for the component contains a <componenttype> element value of APPLICATION_COMPONENT_FACTORY_COMPONENT.
4. The domain profile for the component is the file referenced by an element stated in item 2 and the SPD, SCD and PRF(s) it references.

3.8 Common Context for requirements associated with an ApplicationComponentFactoryComponent

The following assertions apply to the "Context" section of all test procedures for requirements associated with ApplicationComponentFactoryComponent.

1. The ApplicationComponentFactoryComponent is a component part of an application under test.
2. The ApplicationComponentFactoryComponent is a BaseFactoryComponent.
3. Context Group 3.7: Common Context for requirements associated with BaseFactoryComponent.

3.9 Common Context for requirements associated with a component which implements the ComponentFactory interface

The following assertions apply to the "Context" section of all test procedures for requirements associated with components implementing the ComponentFactory IDL interface.

1. The ApplicationComponentFactoryComponent implementing the ComponentFactory interface is a component part of an application under test.
2. Context Group 3.8: Common Context for requirements associated with an ApplicationComponentFactoryComponent.

4 Common Precondition Groups

4.1 Common Precondition for test procedures involving XML

The following statements apply to the "Preconditions" section of all test procedures for which XML files must be read.

1. The XML files should be in their final state and not require further modifications other than those needed to port the application to the target operating environment.
2. The domain profile files should reside (either on the target or an offline directory) in the same relative directory structure as defined in the XML.

4.2 Common Precondition for test procedures involving parsing of XML files

The following statements apply to the "Preconditions" section of all test procedures for which XML files must be parsed.

1. Precondition Group 4.1: Common Precondition for test procedures involving XML.
2. Requirement SCA463 was verified with the domain profile file (SPD/SAD and the XML files that it references).

4.3 Common Precondition for test procedures requiring an application to be created

The following statements apply to the “Preconditions” section of all test procedures for which an instance of an application must have been created.

1. The application under test has been installed on the test platform.
2. An instance of the application under test can be created and a reference to the created application is available to the test procedure.

4.4 Common Precondition for test procedures involving execution of a ManageableApplicationComponent

The following statements apply to the “Preconditions” section of all test procedures for which a ManageableApplicationComponent instance is created.

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a ManageableApplicationComponent implements the Component Registration UoF and that requirement SCA82 was verified.

4.5 Common Precondition for test procedures involving execution of a BaseFactoryComponent

The following statements apply to the “Preconditions” section of all test procedures for which a BaseFactoryComponent instance is created.

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseFactoryComponent implements the Component Registration UoF and that requirement SCA82 was verified.

5 Test Procedures

The model representation of an SCA application (waveform) is provided in Figure 1.

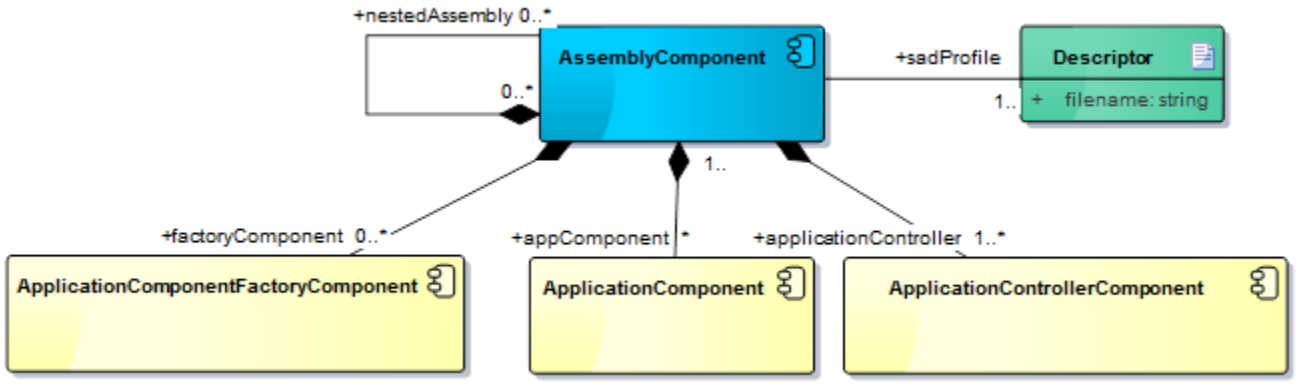

Figure 1: SCA application model

ApplicationComponents, ManageableApplicationComponents, and ApplicationControllerComponents have additional relationships, which are shown in Figure 2.

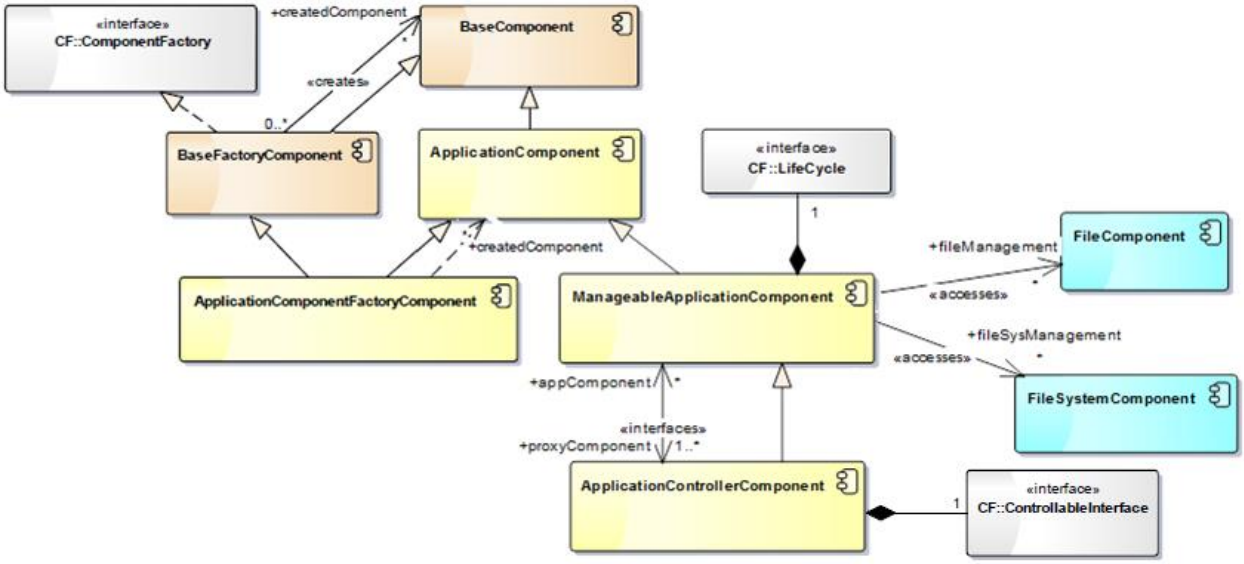

Figure 2: ApplicationComponent, ApplicationComponentFactoryComponent, ManageableApplicationComponent, and ApplicationControllerComponent relationships

As a result, a waveform that is submitted for SCA verification will be composed of some combination of the following components:

- ApplicationControllerComponent
- ManageableApplicationComponent
- ApplicationComponentFactoryComponent
- ApplicationComponent
- AssemblyComponent

The following subsections specify the applicable requirements for each type of component contained within the waveform and their test procedures. Test procedures for the mandatory requirements are grouped together while those for other requirements are grouped by Unit of Functionality. Test procedures may reference other test procedures (e.g. Execute the test procedures of the following requirements: SCA173, SCA457, SCA551, SCA506 of the application under test relative to its incorporated Units of Functionality). The BaseComponent Test Procedures subsection contains test procedures for requirements that are used also by other, non-application, SCA components.

The majority of the time, the test procedures within this specification are written such that they can be used to verify all implementations contained within an application component. The determination regarding whether all of a component's implementations or only those applicable to the target execution platform should be subject to verification is left to the discretion of the testing authority.

5.1 ApplicationControllerComponent Test Procedures

5.1.1 Mandatory

5.1.1.1 Requirement under Test: SCA175

Requirement Text: An ApplicationControllerComponent shall fulfill the ManageableApplicationComponent requirements.

Test Plan Objective/Summary: Ensure that an ApplicationControllerComponent fulfills all requirements of a ManageableApplicationComponent by invoking all test procedures associated with ManageableApplicationComponent for that ApplicationControllerComponent.

Context:

1. Context Group 3.6: Common Context for requirements associated with ApplicationControllerComponent.

Preconditions: N/A

Test Procedure:

Table 1: Steps to execute Test Procedure for SCA175

Step	Action	Expected Result
1	Execute the test procedures of the following requirements: SCA455, SCA456, SCA520, SCA166, SCA167, SCA550 and SCA168 of the application under test relative to its incorporated Units of Functionality.	The application is validated to have an implementation that complies with the set of ManageableApplicationComponent requirements identified by its Units of Functionality.

Postconditions: N/A

Test Plan Verification Method: N/A (The verification method of individual test procedures will apply)

Test Plan Result Category: N/A (The result category of individual test procedures will apply)

5.1.1.2 Requirement under Test: SCA176

Requirement Text: An ApplicationControllerComponent shall realize the ControllableInterface interface.

Test Plan Objective/Summary: Ensure an ApplicationControllerComponent inherits the ControllableInterface as per realization of the ControllableInterface IDL interface. The test procedures obtain an instance of the component and narrow it to CF::ControllableInterface to validate the requirement.

Context:

1. Context Group 3.6: Common Context for requirements associated with ApplicationControllerComponent.
2. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
3. Context Group 3.2: Common Context related to the validation of an application component.
4. For a nested application, only the parent app/root app will be addressed by this procedure. The procedure will have to be seperately enforced for each.

Preconditions:

1. Precondition Group 4.4: Common Precondition for test procedures involving execution of a ManageableApplicationComponent (an ApplicationControllerComponent is a ManageableApplicationComponent).

Test Procedure:

Table 2: Steps to execute Test Procedure for SCA176

Step	Action	Expected Result
1	Obtain the ComponentType struct of the ApplicationManager instance of the application under test.	The ComponentType struct is obtained.
2	Locate the <assemblycontroller> element within the SAD.	The <assemblycontroller> element is found within the SAD.
3	Retrieve the id of the next instantiation reference within the <assemblycontroller> element.	The id of the next instantiation reference is obtained or the verification will terminate.
4	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2.
5	Narrow the componentObject field of the ComponentType reference to the CF:Controllable interface.	The result of the CORBA::is_nil operation on the narrowed componentObject field is false.
6	Repeat steps 3-5 until no more instantiations are found within the <assemblycontroller> element.	The next instantiation will be evaluated or the verification will end.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.1.1.3 Requirement under Test: SCA496

The test procedure for SCA496 is included within the procedure in section "5.6.1.1 Requirement under Test: SCA155, SCA503, SCA494, SCA495, SCA496, SCA500".

5.1.2 Channel Extension UoF

5.1.2.1 Requirement under Test: SCA500

The test procedure for SCA500 is included within the procedure in section "5.6.1.1 Requirement under Test: SCA155, SCA503, SCA494, SCA495, SCA496, SCA500".

5.2 ManageableApplicationComponent Test Procedures

5.2.1 Mandatory

5.2.1.1 Requirement under Test: SCA166

Requirement Text: A ManageableApplicationComponent shall perform file access through the FileSystem and File interfaces.

Test Plan Objective/Summary: Ensure a ManageableApplicationComponent does not invoke any of the POSIX file access functions stated in the context of the test procedure.

Context:

1. Context Group 3.5: Common Context for requirements associated with ManageableApplicationComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. This test procedure precludes the use of POSIX functions, RTOS-specific functions, or any other non-SCA functions which perform file or directory I/O.
4. This test procedure will not preclude I/O function calls if they apply to other types of I/O than files (Sockets, Serial Port, Shared Memory, etc).
5. The following POSIX file / directory functions are not allowed by SCA166. This also includes the reentrant variants of these functions, that is, one of the following function names appended with "_r", for example readdir_r(), if applicable.
    a. File System
        i. mount(), umount()
    b. Common File / Directory Functions
        i. mknod(), mknodat()
        ii. ftruncate(), truncate()
        iii. rename(), renameat()
        iv. remove()
        v. umask()
        vi. fseek(), fseeko(), lseek(), seek(), fgetpos(), ftell(), ftello(), tell(), fsetpos(), rewind()
        vii. fstat(), fstatat(), stat(), lstat()
        viii. fchmod(), chmod()
        ix. fchown(), fchownat(), chown()
        x. ferror(), clearer()
        xi. fflush()
        xii. feof()
        xiii. flock(), lock(), flockfile(), ftrylockfile(), funlockfile()
        xiv. fcntl(), dup(), dup2()
        xv. fsync(), sync(), fdatasync(), aio_fsync()
        xvi. lio_listio()
        xvii. fileno()
        xviii. aio_error(), aio_return(), aio_suspend(), aio_cancel()
        xix. tempnam(),tmpnam(), tmpfile(), mkstemp(), mkostemp(), mkstemps(), mkostemps()

c. File Specific Functions
    i. fopen(), fdopen(), freopen(), open(), openat(), creat()
    ii. fread(), read(), readv(), fgetc(), aio_read(),getline(), getdelim()
    iii. fscanf(), scanf(), vfscanf(), and the wide-character code variants (for example, fwscanf())
    iv. fgetc(), getc(), getchar(), ungetc() and the wide-character code variants (for example fgetwc())
    v. fgets(), gets() and the wide-character code variants (for example fgetws())
    vi. fwrite(), write(), fwritev(), aio_write()
    vii. fputc(), putc(), putc_unlocked(), putchar(), putchar_unlocked()
    viii. fputs(), puts()
    ix. fprintf(), printf(), dprintf(), vdprintf(), vfprintf(), vprintf(), vsprintf(), vsnprintf()
    x. fclose(), close()
d. Directory File Functions
    i. fchdir(), chdir()
    ii. mkdir(), mkdirat()
    iii. opendir(), fdopendir()
    iv. closedir()
    v. rmdir()
    vi. rmtree()
e. Directory Entry Functions
    i. readdir()
    ii. flink(), link(), linkat(), readlink(), readlinkat(), symlink(), symlinkat()
    iii. unlink(), unlinkat()

Preconditions:

1. The application developer provides the source file name for the ManageableApplicationComponent as identified by item 2 in Context Group 3.5 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the ManageableApplicationComponent.

Test Procedure:

Table 3: Steps to execute Test Procedure for SCA166

Step	Action	Expected Result
1	Locate the SAD file of the application and open it.	The SAD file of the application is found and is opened.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 4 if the element being evaluated is a <componentplacement> or return to step 1 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain type of a component placement"	See sub-procedure 6.3.
6	Verify that the value of the <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT.	The value of <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT or go back to step 3.
7	Determine the source file names associated with the SCD for the ManageableApplicationComponent (see Preconditions for this information).	The source file names associated with the SCD for the ManageableApplicationComponent is determined.
8	Search the source files of the ManageableApplicationComponent for non-SCA functions which perform file or directory I/O. See "Content" for non-SCA function names. Go to step 3.	No non-SCA functions are located which perform file or directory I/O. Go to step 3.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Result Category: N/A

5.2.1.2 Requirement under Test: SCA167

Requirement Text: All ManageableApplicationComponent processes shall have a handler registered for the AEP SIGQUIT signal.

Test Plan Objective/Summary: Ensure a ManageableApplicationComponent configures a function to handle the SIGQUIT signal.

Context:

1. Context Group 3.5: Common Context for requirements associated with ManageableApplicationComponent.
2. The function which handles the POSIX SIGQUIT signal can be configured by invoking either the sigaction() or signal() function. Both functions are mandatory for the "AEP" profile in SCA 4.1 Appendix B. Other profiles in SCA 4.1 Appendix B do not support these functions.
3. The POSIX standard is maintained by http://opengroup.org/. Per the "Application Usage" section in the sigaction() manual page on the opengroup.org website, "The sigaction() function supersedes the signal() function, and should be used in preference."
4. This test procedure assumes the sigaction() function is invoked to configure a function to handle the SIGQUIT signal.
5. The sigaction function signature is "int sigaction(int sig, const struct sigaction *restrict act, struct sigaction *restrict oact)". sigaction() parameters and fields in the "struct sigaction act" parameter not referenced by the test procedure are outside the scope of requirement SCA167.

Preconditions:

1. The application developer provides the source file name for the ManageableApplicationComponent as identified by item 2 in Context Group 3.5 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the ManageableApplicationComponent.

Test Procedure:

Table 4: Steps to execute Test Procedure for SCA167

Step	Action	Expected Result
1	Locate the SAD file of the application and open it.	The SAD file of the application is found and is opened.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 4 if the element being evaluated is a <componentplacement> or return to step 1 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain type of a component placement"	See sub-procedure 6.3.
6	Verify that the value of the <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT.	The value of <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT or go back to step 3.
7	Determine the source file names associated with the SCD for the ManageableApplicationComponent (see Preconditions for this information).	The source file names associated with the SCD for the ManageableApplicationComponent is determined.
8	Search the source files of the ManageableApplicationComponent for non-SCA functions which perform file or directory I/O. See "Content" for non-SCA function names. Go to step 3.	No non-SCA functions are located which perform file or directory I/O. Go to step 3.
9	In the source files which implement the ManageableApplicationComponent, locate the invocation of the sigaction() function.	The invocation of the sigaction() function is located.
10	Verify the "sig" parameter value passed to sigaction() is SIGQUIT and the SIGQUIT definition used is from signal.h.	The "sig" parameter value passed to sigaction() is SIGQUIT and the SIGQUIT definition used is from signal.h.
11	Verify the "act" parameter value passed to sigaction() is a variable of data type "struct sigaction".	The "act" parameter value passed to sigaction() is a variable of data type "struct sigaction".
12	Verify the "sa_handler" field of the "act" parameter contains the name of a defined function.	The "sa_handler" field of the "act" parameter contains the name of a defined function.
13	Go to step 3	Go to step 3.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Result Category: N/A

5.2.1.3 Requirement under Test: SCA169

Requirement Text: Each ApplicationComponent shall be accompanied by an SPD file per section 3.1.3.6.

No test is provided since the test procedure for SCA427 addresses this requirement.

5.2.1.4 Requirement under Test: SCA455, SCA82

Requirement Text:

SCA455: Each ManageableApplicationComponent shall support the mandatory Component Identifier execute parameter as described in section 3.1.3.3.1.3.5.1, in addition to their user-defined execute properties in the component's SPD.

SCA82: A ManageableApplicationComponent shall register via the *ComponentRegistry::registerComponent* operation when a COMPONENT_REGISTRY_IOR parameter is supplied.

Test Plan Objective/Summary: Ensure a ManageableApplicationComponent supplies the component identifier provided by a mandatory executable parameter into its ComponentType structure (SCA455). Ensure a ManageableApplicationComponent registers to the ComponentRegistry interface supplied through the COMPONENT_REGISTRY_IOR parameter (SCA82). The ComponentType structure returned by an application contains a specializedInfo field identified by an id of COMPONENTS_ID and a type CF::Components. One element of the CF::Components must have an identifier equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.

Context:

1. Context Group 3.5: Common Context for requirements associated with ManageableApplicationComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
4. The portion of the requirement "in addition to their user-defined execute properties in the component's SPD" will be addressed in the validation of SCA456.

Preconditions:

1. Precondition Group 4.4: Common Precondition for test procedures involving execution of a ManageableApplicationComponent.

Test Procedure:

Table 5: Steps to execute Test Procedure for SCA455, SCA82

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Identify the next <componentplacement> element within the <partitioning> element.	The location of the <componentplacement> element is identified or the verification will terminate.
4	Perform steps defined in sub-procedure "Obtain type of a component placement"	See sub-procedure 6.3.
5	Verify that the value of the <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT.	The value of <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT or go back to step 3.
6	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
7	Determine that the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contain a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.
8	Repeat steps 6-7 until no more <componentinstantiation> elements are found within the <componentplacement> element, otherwise go to step 3.	The next componentinstantiation will be evaluated or the verification will go to step 3.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.2.1.5 Requirement under Test: SCA456

Requirement Text: Each executable ManageableApplicationComponent shall accept executable parameters as specified in section 3.1.3.4.1.6.5.1.3 (ExecutableInterface::execute).

Test Plan Objective/Summary: Ensure the executable file of a ManageableApplicationComponent supports all executable parameters specified in the various PRF files associated with the component. The test procedure verifies that the source code of the component performs a search in the input parameter of the entry point function (e.g. argv argument of the main() function) for each executable parameter property id specified in a PRF file for the component and stores the property value associated with each executable parameter.

Context:

1. Context Group 3.5: Common Context for requirements associated with ManageableApplicationComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. The property ID for an executable parameter is defined in one of the PRF files for the ManageableApplicationComponent as a <simple> element with a <kind> subelement having a "type" attribute value of "execparam".
4. When the application is created, a ManageableApplicationComponent may be deployed as a process (executable file) or as an execution thread, depending on the value of the type attribute of the code element of the SPD
    a. process: When the ExecutableInterface::execute() operation is invoked to load and execute the component, it passes the executable parameters to the component as a set of property ID / value pairs via the POSIX "argv" interface, starting in elements argv[1] and argv[2] for the first pair and sequential argv indices for subsequent pairs.
    b. execution thread: When the ExecutableInterface::execute() operation is invoked to load and execute the component, it passes the executable parameters to the component as a set of property ID / value pairs via the POSIX "argv" interface, starting in elements argv[1] and argv[2] for the first pair and sequential argv indices for subsequent pairs.. The executable parameters are passed via the "void *arg" parameter in the pthread_create() function.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. The application developer provides the source file name for the ManageableApplicationComponent as identified by item 2 in Context Group 3.5 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the ManageableApplicationComponent.

Test Procedure: Perform the following for each ManageableApplicationComponent

Table 6: Steps to execute Test Procedure for SCA456

Step	Action	Expected Result
1	Locate the SPD file for the ManageableApplicationComponent (see Context).	The SPD file for the ManageableApplicationComponent is found.

Step	Action	Expected Result
2	Locate the SCD file for the ManageableApplicationComponent, which is defined in the <descriptor> element in the SPD.	The SCD file for the ManageableApplicationComponent is found.
3	Locate all the PRF files for the ManageableApplicationComponent which may be referenced by one or more of the following: 1. In the SPD, in the <propertyfile> in the <softpkg> . 2. In the SPD, in the <propertyfile> in the <implementation>. 3. In the SCD, in the <propertyfile> in the <softwarecomponent>.	All the PRF files for the ManageableApplicationComponent are found.
4	Compile a list of all property IDs of "kindtype" = "execparam" (see Context) from all PRFs for the ManageableApplicationComponent	A list of all property IDs of "kindtype" = "execparam" for the ManageableApplicationComponent is compiled.
5	If the ManageableApplicationComponent is implemented as an execution thread (see Context).	The ManageableApplicationComponent is implemented as an (a) execution thread (continue with step 6) or (b) executable file (go to step 9).
6	Locate the source file containing the main() function (or a function it invokes) or the entry point function definition, which instantiates the ManageableApplicationComponent.	The source file containing the main() function or the entry point function definition, which instantiates the ManageableApplicationComponent is found.
7	Verify the "argv" value is parsed as a sequence of character arrays, with the first property ID / value pair in argv[1] and argv[2], and subsequent property ID / value pairs in sequentially higher indices of argv.	The function parses the "argv" value as a sequence of character arrays, with the first property ID / value pair in argv[1] and argv[2], and subsequent property ID / value pairs in sequentially higher indices of argv.
8	Verify the function it invokes, for each property ID / pair passed in compares each property ID to a string literal for each a <simple> element with "kindtype" = "execparam" in a PRF for the ManageableApplication Component and when a match is found.	The function it invokes, for each property ID / pair passed in compares each property ID to a string literal for each a <simple> element with "kindtype" = "execparam" in a PRF for the ManageableApplication Component and when a match is found.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Result Category: N/A

5.2.1.6 Requirement under Test: SCA520

Requirement Text: A ManageableApplicationComponent shall fulfill the ApplicationComponent requirements.

Test Plan Objective/Summary: Ensure that a ManageableApplicationComponent fulfills all requirements of an ApplicationComponent by invoking all test procedures associated with an ApplicationComponent for that ManageableApplicationComponent.

Context:

1. Context Group 3.5: Common Context for requirements associated with ManageableApplicationComponent.

Preconditions: N/A

Test Procedure:

Table 7: Steps to execute Test Procedure for SCA520

Step	Action	Expected Result
1	Execute the test procedures of the following requirements: SCA551, SCA173, SCA457 and SCA506 of the application under test relative to its incorporated Units of Functionality.	The application is validated to have an implementation that complies with the set of ApplicationComponent requirements identified by its Units of Functionality.

Postconditions: N/A

Test Plan Verification Method: N/A (The verification method of individual test procedures will apply).

Test Plan Result Category: N/A (The result category of individual test procedures will apply).

5.2.1.7 Requirement under Test: SCA550

Requirement Text: A ManageableApplicationComponent shall realize the LifeCycle interface.

Test Plan Objective/Summary: Ensure a ManageableApplicationComponent inherits the LifeCycle interface as per realization of the LifeCycle IDL interface. The test procedure obtains an instance of the component and narrows it to CF::LifeCycle to validate the requirement.

Context:

1. Context Group 3.5: Common Context for requirements associated with ManageableApplicationComponent.

2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.4: Common Precondition for test procedures involving execution of a ManageableApplicationComponent.

Test Procedure:

Table 8: Steps to execute Test Procedure for SCA550

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain type of a component placement"	See sub-procedure 6.3.
6	Verify that the value of the <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT.	The value of <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Determine that the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contain a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.
9	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::LifeCycle interface.	The componentObject can be narrowed to CF::LifeCycle interface.
11	Return to step 3.	The next componentwill be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.2.2 Interrogable UoF

5.2.2.1 Requirement under Test: SCA168

Requirement Text: Each executable ManageableApplicationComponent shall set its identifier attribute using the Component Identifier execute parameter.

Test Plan Objective/Summary: Ensure a ManageableApplicationComponent has a component identifier attribute value that is equivalent to that stored within the component's ComponentType structure. The ComponentType structure returned by an application contains a specializedInfo field identified by an id of COMPONENTS_ID and a type CF::Components. One element of the CF::Components must have an identifier equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component. The test procedure obtains an instance of the component and narrows it to CF::ComponentIdentifier to validate the requirement.

Context:

1. Context Group 3.5: Common Context for requirements associated with ManageableApplicationComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.4: Common Precondition for test procedures involving execution of a ManageableApplicationComponent.
2. This test procedure requires that the requirement SCA455 has been verified prior.

Test Procedure:

Table 9: Steps to execute Test Procedure for SCA168

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ApplicationManager reference is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain type of a component placement"	See sub-procedure 6.3.

Step	Action	Expected Result
6	Verify that the value of the <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT.	The value of <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT or go back to step 3.
7	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
8	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the ComponentIdentifier interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the ComponentIdentifier interface IDL repository ID or go back to step 3.
9	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
10	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2.
11	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
12	Narrow the componentObject to the CF::ComponentIdentifier interface.	The componentObject can be narrowed to CF::ComponentIdentifier interface.
13	Retrieve the identifier attribute from the CF::ComponentIdentifier interface and compare it against the value that corresponds to the COMPONENTS_ID id within the specializedInfo field of the application's ComponentType.	The compared values will be equal or the verification procedure will fail.
14	Repeat steps 9-13 until no more <componentinstantiation> elements are found within the <componentplacement> element, otherwise go to step 3.	The next componentinstantiation will be evaluated or the verification will go to step 3.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.2.3 Component Registration UoF

5.2.3.1 Requirement under Test: SCA82

The test procedure for SCA82 is included within the procedure in section "5.2.1.4 Requirement under Test: SCA455, SCA82".

5.3 ApplicationComponentFactoryComponent Test Procedures

5.3.1 Mandatory

5.3.1.1 Requirement under Test: SCA415

The test procedure for SCA415 is included within the procedure in section "5.7.1.1 Requirement under Test: SCA386, SCA387, SCA388, SCA415".

5.3.1.2 Requirement under Test: SCA521

Requirement Text: An ApplicationComponentFactoryComponent shall fulfill the BaseFactoryComponent requirements.

Test Plan Objective/Summary: Ensure that an ApplicationComponentFactoryComponent fulfills all requirements of a BaseFactoryComponent by invoking all test procedures associated with BaseFactoryComponent for that ApplicationComponentFactoryComponent.

Context:

1. Context Group 3.8: Common Context for requirements associated with an ApplicationComponentFactoryComponent.

Preconditions: N/A

Test Procedure:

Table 10: Steps to execute Test Procedure for SCA521

Step	Action	Expected Result
1	Execute the test procedures of the following requirements: SCA386, SCA387, SCA388, SCA389, SCA540, SCA541, SCA574, SCA413, SCA414, SCA549 of the application under test relative to its incorporated Units of Functionality.	The application is validated to have an implementation that complies with the set of BaseFactoryComponent requirements identified by its Units of Functionality.

Postconditions: N/A

Test Plan Verification Method: N/A (The verification method of individual test procedures will apply)

Test Plan Result Category: N/A (The result category of individual test procedures will apply)

5.3.1.3 Requirement under Test: SCA522

Requirement Text: An ApplicationComponentFactoryComponent shall fulfill the ApplicationComponent requirements.

Test Plan Objective/Summary: Ensure that an ApplicationComponentFactoryComponent fulfills all requirements of an ApplicationComponent by invoking all test procedures associated with ApplicationComponent for that ApplicationComponentFactoryComponent.

Context:

1. Context Group 3.8: Common Context for requirements associated with an ApplicationComponentFactoryComponent.

Preconditions: N/A

Test Procedure:

Table 11: Steps to execute Test Procedure for SCA522

Step	Action	Expected Result
1	Execute the test procedures of the following requirements: SCA173, SCA457, SCA551, SCA506 of the application under test relative to its incorporated Units of Functionality.	The application is validated to have an implementation that complies with the set of ApplicationComponent requirements identified by its Units of Functionality.

Postconditions: N/A

Test Plan Verification Method: N/A (The verification method of individual test procedures will apply)

Test Plan Result Category: N/A (The result category of individual test procedures will apply)

5.4 ApplicationComponent Test Procedures

5.4.1 Mandatory

5.4.1.1 Requirement under Test: SCA551

Requirement Text: An ApplicationComponent shall fulfill the BaseComponent requirements.

Test Plan Objective/Summary: Ensure that an ApplicationComponent fulfills all requirements of a BaseComponent by invoking all test procedures associated with BaseComponent for that ApplicationComponent.

Context:

1. The ApplicationComponent is a part of an application under test.
2. A component is identified by the <componentplacement> element in the SAD of the application under test.

Preconditions: N/A

Test Procedure:

Table 12: Steps to execute Test Procedure for SCA551

Step	Action	Expected Result
1	Execute the test procedures of the following requirements: SCA427, SCA430, SCA548, SCA463, SCA501, SCA502, SCA503, SCA494, SCA495, SCA420, SCA421, SCA423, SCA429, SCA545, SCA26, SCA28, SCA29, SCA30, SCA31, SCA432, SCA15, SCA518, SCA16, SCA17, SCA18, SCA433, SCA32, SCA33, SCA34, SCA36, SCA37, SCA547, SCA7, SCA519, SCA8, SCA10, SCA11, SCA12, SCA13, SCA14, SCA424, SCA425, SCA444, SCA426, SCA6, SCA428, SCA546, SCA19, SCA21, SCA23, SCA24, SCA25 of the application under test relative to its incorporated Units of Functionality.	The application is validated to have an implementation that complies with the set of BaseComponent requirements identified by its Units of Functionality.

Postconditions: N/A

Test Plan Verification Method: N/A (The verification method of individual test procedures will apply)

Test Plan Result Category: N/A (The result category of individual test procedures will apply)

5.4.2 AEP Compliant UoF

5.4.2.1 Requirement under Test: SCA173

Requirement Text: An ApplicationComponent shall be limited to using the mandatory OS services designated in Appendix B as specified in the SPD.

Test Plan Objective/Summary: Ensure the ApplicationComponent does not invoke any of the non-required AEP functions as stated in the context of the test procedure.

Context:

1. The SPD and “source files” referenced in this test case apply to each implemented ApplicationComponent.
2. A “mandatory OS service” is a POSIX function referenced in an Appendix B table with “MAN” under the “AEP”, “LwAEP” or “ULwAEP” columns.
3. Each SPD implementation element “aepcompliance” attribute specifies which AEP profile the ApplicationComponent conforms with. The possible values of the “aepcompliance” attribute are “aep_compliant”, “lw_aep_compliant”, “ulw_aep_compliant “ or “aep_non_compliant” with a default value of “aep_compliant”. There may be multiple implementation elements and each element may specify a different AEP profile.

4. This requirement does not apply to an ApplicationComponent which supports the "aep_non_compliant" AEP profile.
5. For an ApplicationComponent to comply with this requirement, its source files may only invoke mandatory OS services for its declared AEP profile.
6. A source file which invokes OS services may be conformant with an AEP profile via conditional compilation or by implementing each AEP profile in a separate source file without using conditional compilation.
7. If this requirement is verified by Source code inspection the following must occur:
   7.1. Each source file which invokes OS services is inspected to ensure only the mandatory OS services defined by the declared AEP profile are used.
   7.2. When a source file can be used to conform with multiple AEP profiles via conditional compilation controlled by preprocessor directives, verification is performed in accordance with each profile, only evaluating the code that is available for compilation after preprocessing the directives associated with that profile.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. The developer specifies the source file names used to build the ApplicationComponent executable.
3. If the SPD contains multiple implementation elements such that more than one AEP profile is realized by the source file(s)
   3.1. The developer identifies which AEP profile(s) is (are) to be evaluated for AEP compliance
   3.2. If multiple AEP profiles are implemented via:
      3.2.1. a single source file using conditional compilation, the developer identifies the C preprocessor symbol(s) that control conditional compilation that determines which AEP profile is implemented.
      3.2.2. a separate source file implementing each profile without using conditional compilation, the developer identifies which source file supports which AEP profile.
4. An RTOS symbol resolution object file exists on the test platform for each AEP profile. This may be a fully functional implementation of the mandatory OS services for a specific AEP profile or may only be the AEP constants and function signatures, and if applicable, a default return value in the function body, for each of the mandatory OS services.
5. An SCA CF function and symbol resolution object file may exist on the test platform. This may be a fully functional SCA CF implementation generated by a CORBA IDL to CPP compiler or only the SCA constants and function signatures, and if applicable, a default return value in the function body, for each of the CF interfaces.
6. A CORBA ORB function and symbol resolution object file may exist on the test platform. This may be a fully functional CORBA ORB implementation or only the CORBA constants and function signatures, and if applicable, a default return value in the function body, for each of the CORBA interfaces.

Test Procedure:

Table 13: Steps to execute Test Procedure for SCA173

Step	Action	Expected Result
1	Locate all the ApplicationComponents to which the test procedure applies.	A list of ApplicationComponents to which the test procedure applies is identified.
2	Compile the source files of the next ApplicationComponent in the list.	The source files are successfully compiled and the resulting object files are saved on the test platform.
3	Link the ApplicationComponent's object files with the RTOS symbol resolution file of the target AEP profile, and if present, the CF and the CORBA function symbol resolution object files.	The linker reports all external function references that it cannot resolve, which may be non-compliant OS services or middleware functions. If the CF and CORBA function symbol resolution object files are not present, the linker will report external CF and CORBA function references it cannot resolve.
4	Evaluate the linker output error messages to determine if they reference non-required OS services for the AEP profile being evaluated for compliance. Disregard linker error messages that do not reference with OS services in Appendix B, such as SCA CF interfaces and constants, CORBA functions and constants.	If there are no linker error messages references non-required OS services for the AEP profile being evaluated for compliance, the ApplicationComponent conforms to the AEP profile.
5	Repeat steps 2-4 until all AEP profiles for the ApplicationComponent to be inspected for AEP compliance have been evaluated.	The next AEP profile for the ApplicationComponent is evaluated or continue with step 6.
6	Continue with step 2 for the next ApplicationComponent to be evaluated for AEP compliance.	The next ApplicationComponent for AEP compliance is evaluated or the test terminates.

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Necessary [Ref1]

If the test passes, the ApplicationComponent does not invoke any of the non-required AEP functions stated in the test procedure. The test does not confirm the ApplicationComponent invokes only mandatory AEP functions.

5.5 AssemblyComponent Test Procedures

5.5.1 Mandatory

5.5.1.1 Requirement under Test: SCA155

The test procedure for SCA155 is included within the procedure in section "5.6.1.1 Requirement under Test: SCA155, SCA503, SCA494, SCA495, SCA496, SCA500".

5.5.1.2 Requirement under Test: SCA156

Requirement Text: An AssemblyComponent shall have at least one ApplicationControllerComponent.

Test Plan Objective/Summary: Ensure that the SAD contains an <assemblycontroller> element with an instantiation reference defined in the SAD.

Context:

1. Context Group 3.6: Common Context for requirements associated with ApplicationControllerComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
4. For a nested application, only the parent app/root app need be addressed by this procedure. The procedure may be performed for each nested SAD file submitted with an application.

Preconditions:

1. Precondition Group 4.4: Common Precondition for test procedures involving execution of a ManageableApplicationComponent (an ApplicationControllerComponent is a ManageableApplicationComponent).

Test Procedure:

Table 14: Steps to execute Test Procedure for SCA156

Step	Action	Expected Result
1	Locate the <assemblycontroller> element within the SAD.	The <assemblycontroller> element is found within the SAD.
2	Retrieve the id of the next componentinstantion or assemblyinstantiation reference within the <assemblycontroller> element.	The id of the next instantiation reference is obtained or the verification will terminate.
3	Locate in the SAD a <componentinstantiation> element or an <assemblyinstantiation> element with a value of the ID attribute equal to the id retrieved in step 2.	A matching <componentinstantiation> element or <assemblyinstantiation> element is found and go to step 2.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.5.1.3 Requirement under Test: SCA457

Requirement Text: An ApplicationComponent shall be limited to using transfer mechanisms features specified in Appendix E for the specific platform technology implemented.

Context:

1. An ApplicationComponent is part of an application under test.
2. An ApplicationComponent is a BaseComponent.
3. Context Group 3.2: Common Context related to the validation of an application component.
4. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
5. This procedure is focused on CORBA as the transport mechanism. If additional transport mechanisms are added to the SCA, this procedure would need to be adapted accordingly.

Preconditions:

1. The application developer provides the source file name for the ApplicationComponent as identified by item 2 in Context Group 3.4 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the ApplicationComponent.
2. The application developer indicates the CORBA profile for the verification.

Test Procedure:

Table 15: Steps to execute Test Procedure for SCA457

Step	Action	Expected Result
1	Locate the SAD file of the application and open it.	The SAD file of the application is found and is opened.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	1. Go to step 4 if the element being evaluated is a <componentplacement> or return to step 1 and proceed with the file identified in the <assemblyplacement>.	1. The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain type of a component placement"	See sub-procedure 6.3.
6	Verify that the value of the <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT.	The value of <componenttype> element is MANAGEABLE_APPLICATION_COMPONENT or go back to step 3.
7	Determine the source file names associated with the SCD for the ManageableApplicationComponent (see Preconditions for this information).	The source file names associated with the SCD for the ManageableApplicationComponent is determined.
8	Search the ApplicationComponent's source code for uses of CORBA. Note the most restrictive CORBA profile (Full, Lightweight, or Ultralightweight) that the source code adheres to.	The ApplicationComponent does not declare using a CORBA profile more restrictive than the one found to be used (e.g. declared Lightweight and found Full).
9	Search the ApplicationComponent's source code for other transfer mechanisms. Go to step 3.	No other transfer mechanism is found. Go to step 3.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Verification Status: N/A

5.6 BaseComponent Test Procedures

5.6.1 Mandatory

5.6.1.1 Requirement under Test: SCA155, SCA503, SCA494, SCA495, SCA496, SCA500

Requirement Text:

SCA155: An AssemblyComponent shall be accompanied by the appropriate Domain Profile files per section 3.1.3.6.

SCA503: A Software Package Descriptor file shall have a ".spd.xml" extension.

SCA494: A Properties Descriptor shall have a ".prf.xml" extension.

SCA495: A Software Component Descriptor file shall have a ".scd.xml" extension.

SCA496: A Software Assembly Descriptor file shall have a ".sad.xml" extension.

SCA500: An Application Deployment Descriptor file shall have an ".add.xml" extension.

Test Plan Objective/Summary: Ensure a Domain Profile file name has a file extension as specified by the associated requirement.

Context:

1. The SAD, SPD, SCD or PRF file is one in the domain file of an application component of an application under test.
2. Context Group 3.2: Common Context related to the validation of an application component.

Preconditions:

1. Precondition Group 4.1: Common Precondition for test procedures involving XML.
2. Requirement 502 should be verified with the domain profile files of the application under test.
3. The application developer provides the source file name for the AssemblyComponent SAD. This is an unnecessary precondition if the verifier can determine the name of the SAD.

Test Procedure:

Table 16: Steps to execute Test Procedure for SCA503, SCA494, SCA495, SCA496, SCA500

Step	Action	Expected Result
1	Locate the identified SAD file of the AssemblyComponent (SCA155).	The SAD file of the application is found and can be open.
2	Identify the document type by reading the !DOCTYPE element.	The document type is identified.
3	Recursively traverse the domain profile files starting from the root SAD and construct a list of the unique domain profile files referenced by the application.	All the application's referenced domain profile files have been collected.
4	For each file collected in Step 3, open the file and identify the document type and verify the extension as follows: a. If the document type is "softwareassembly", verify that the file extension is ".sad.xml". (SCA496) b. If the document type is "softpkg", verify that the file extension is ".spd.xml". (SCA503) c. If the document type is "softwarecomponent", verify that the file extension is ".scd.xml". (SCA495) d. If the document type is "properties", verify that the file extension is ".prf.xml". (SCA494) e. If the document type is "deploymentprecedence", verify that the file extension is ".add.xml". (SCA500)	The referenced domain profile file extension is validated.

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.1.2 Requirement under Test: SCA427

Requirement Text: A BaseComponent shall be associated with a domain profile file.

Test Plan Objective/Summary: Ensure that an application component, which is a BaseComponent, is really associated with an SPD file since the refid attribute of a componentfileref element used to do the association shall have a valid file path as value. A valid file path is one for which the existence of the file can be verified.

Context:

1. Context Group 3.2: Common Context related to the validation of an application component.
2. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.

Test Procedure:

Table 17: Steps to execute Test Procedure for SCA427

Step	Action	Expected Result
1	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
2	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
3	Return to step 2 if the element being evaluated is a <componentplacement> element or return to step 1 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.1.3 Requirement under Test: SCA430

The test procedure for SCA155 is included within the procedure in section "5.6.2.6 Requirement under Test: SCA13, SCA14, SCA430".

5.6.1.4 Requirement under Test: SCA463

Requirement Text: Domain Profile files shall be compliant to the descriptor files provided in Appendix D.

Test Plan Objective/Summary: Ensure that Domain Profile files are valid with respect to the SCA 4.1 XML descriptor files.

Context:

1. Context Group 3.2: Common Context related to the validation of an application component.

Preconditions:

1. Precondition Group 4.1: Common Precondition for test procedures involving XML.

Test Procedure:

Table 18: Steps to execute Test Procedure for SCA463

Step	Action	Expected Result
1	Read the domain profile files to which this requirement applies with a standard XML parser or tool that has the capability to perform DTD validation (the DTD validation will be enabled).	The domain profile file being read is valid with respect to the applicable DTD.
2	Identify all domain profile files referenced in the current domain profile file not already analyzed.	All referenced domain profile files that have not been analyzed are identified.
3	Perform steps 1-2 for each of the identified files that have not already been analyzed.	N/A

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.1.5 Requirement under Test: SCA501

Requirement Text: DTD files are installed in the domain and shall have ".dtd" as their filename extension.

No test is provided since the DTDs shall be installed on the platform with the OE and this should be an OR requirement.

5.6.1.6 Requirement under Test: SCA502

Requirement Text: All XML files shall have as the first two lines as an XML declaration (?xml) and a document type declaration (!DOCTYPE).

Test Plan Objective/Summary: Ensure the Domain Profile files contain the first two lines as specified by the requirement.

Context:

1. The XML files are the domain profile file of an application and its application components.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. The first two lines are interpreted as omitting white space and comments.

Preconditions:

1. Precondition Group 4.1: Common Precondition for test procedures involving XML.

Test Procedure:

Table 19: Steps to execute Test Procedure for SCA502

Step	Action	Expected Result
1	Locate the SAD file of the application.	The SAD file of the application is found.
2	Open the domain profile file and read the first non-commentary statement.	The test process has the text for the first line of code within the domain profile file.
3	Validate that the text "?xml" is the first string encountered after the leading "<" character.	The required "?xml" definition in the proper location is identified.
4	Validate that the text has a "version" definition within the statement.	The version definition is identified within the statement.
5	Read the second non-commentary xml statement.	The test process has the text for the second line of code within the domain profile file.
6	Validate that the text "!DOCTYPE" is the first string encountered after the leading "<" character	The doctype definition in the proper location is identified.
7	Validate that the text has a "SYSTEM" definition within the statement with a reference to a DTD file that is valid for the Operating Environment.	The file references a DTD that is part of the domain profile which is identified within the statement.
8	Identify all domain profile files referenced in the current domain profile file not already analyzed and validate requirement SCA502 for each identified file.	All referenced domain profile files that have not been validated are evaluated.

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.1.7 Requirement under Test: SCA548

Requirement Text: A BaseComponent shall implement its optional composition relationships via inheritance.

Test Plan Objective/Summary: Ensure that an application component, which is a BaseComponent, inherits from CF interfaces for which it has optional composition relationships. The optional interfaces are specified by the <supportsinterface> elements in the application component SCD file. The test procedure will validate the inheritance relationship by evaluating each interface specified by the <supportsinterface> elements within the SCD.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
2. All optional composition relationships are specified within the BaseComponent SCD <supportsinterface> definition.
3. Context Group 3.2: Common Context related to the validation of an application component.
4. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. One or more <supportsinterface> definitions must exist in the SCD file.
2. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.

Test Procedure:

Table 20: Steps to execute Test Procedure for SCA548

Step	Action	Expected Result
1	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
2	Perform steps defined in sub-procedure “Locate next component or assembly”.	See sub-procedure 6.1.
3	Go to step 4 if the element being evaluated is a <componentplacement> or return to step 1 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
4	Perform steps defined in sub-procedure “Obtain supported interface list”.	See sub-procedure6.5.
5	Identify the next <supportsinterface> element within <componentfeatures> element.	The location of a <supportsinterface> element will be identified or the verification will go back to step 2.
6	Determine if the repid attribute of the <supportsinterface> element is one among the BaseComponent optional composition (CF::ControllableInterface, CF::LifeCycle, CF::ComponentIdentifier, CF::PropertySet, CF::PortAccessor, CF::TestableInterface) and refers to an <interface> element within the SCD.	The repid attribute is one of the BaseComponent optional composition and refers to an <interface> element in the SCD or will go to step 5 again.
7	Repeat step 5-6 until no more <supportsinterface> elements are found within the <componentfeatures> element.	The next supportsinterface will be evaluated or the verification will go to step 8.
8	Obtain an object reference for a component instance associated with the <componentplacement> element used in step 2.	An object reference is obtained.
9	For each interface identified in 6, the component can be widened to the supportsinterface.	The component can be widened to the supportsinterface.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.2 Connectable UoF

5.6.2.1 Requirement under Test: SCA7

Requirement Text: The *connectUsesPorts* operation shall make the connection(s) to the component identified by its input portConnections parameter.

Test Plan Objective/Summary: Ensure that a component uses the object provided via connection.

Context:

1. The BaseComponent is a component part of an application under test.
2. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
3. The test procedure only tests one component and should be repeated for every component of an application.
4. The SAD file using the component under test is used to determine the connections to be established in the test procedure.

Preconditions:

1. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
2. All connections are expected to be already established.

Test Procedure:

Table 21: Steps to execute Test Procedure for SCA7

Step	Action	Expected Result
1	Demonstrate through some specific actions that the provides port provided to the uses port of the component are used.	Evidence that the provides port are used is demonstrated.
2	Use some testing tools specific ways to disconnect all connections.	Connections are successfully disconnected.
3	Using the same actions in Step 1, demonstrate that the provides port which were provided to the uses port of the component are not used.	Evidence that the provides port are disconnected.
4	Use some testing tool specific ways to re-establish all connections.	Connections are successfully established.
5	Using the same actions in Step 1, demonstrate that the provides port provided to the uses port of the component are used.	Evidence that the provides port are used is demonstrated.

Postconditions: N/A

Test Plan Verification Method: Demonstration [Ref1]

Test Plan Result Category: N/A

5.6.2.2 Requirement under Test: SCA8

Requirement Text: The *connectUsesPorts* operation shall raise the InvalidPort exception when the input portConnections parameter provides an invalid connection for the specified port.

Test Plan Objective/Summary: Ensure that an application component, which is a BaseComponent, raises an InvalidPort exception when it is asked to connect something invalid. There are four cases for which the InvalidPort can be raised as specified in the context of this procedure. The test procedure parses the SCD file to determine the ports, obtain an instance of the component, and invokes several times the connectUsesPorts operation with different input arguments to cover the four cases described in the context.

Context:

1. Context Group 3.2: Common Context related to the validation of an application component.
2. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
3. The ports of a BaseComponent are identified by the <ports> element in the SCD of the component under test.
4. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
5. Section 3.1.3.2.1.2.3.6 of the SCA specification describes four cases where the InvalidPort can be raised. The following cases apply for the connectUsesPorts operation:
    a. The provides portReference is invalid (e.g. unable to narrow object reference) or nil object reference.
    b. The connectionId is invalid.
    c. The uses port portName does not exist for the given connectionId.
    d. The port has reached its maximum number of connections and is unable to accept any additional connections.
6. An object implementing an IDL interface not defined in the SCA specification and that doesn't inherit from any SCA interface is provided by the SCA 4.1 Test Platform.

Preconditions:

1. The existence of a capability on the test platform that is able to establish connections through the PortAccessor interface between a uses port component and provides port component.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
4. The domain profile of the BaseComponent includes an SCD file that contains one or more port definitions.

Test Procedure:

Table 22: Steps to execute Test Procedure for SCA8

Step	Action	Expected Result
1	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
2	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
3	Go to step 4 if the element being evaluated is a <componentplacement> or return to step 1 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
4	Open the SPD file indicated by the <componentfile> referenced by the <componentplacement> and locate the <descriptor> element.	The <descriptor> element will be found or the verification will go back to step 2.
5	Determine if the file identified by <localfile> element referenced by the <descriptor> element exists.	The file identified by the <localfile> reference within the <descriptor> subelement exists.
6	Open the SCD file indicated by the <localfile> referenced by the <descriptor> and locate the <componentfeatures> element within the SCD.	The <componentfeatures> element is found within the SCD.
7	Locate the <ports> element within <componentfeatures> element.	The location of the <ports> element is identified.
8	Identify all the <uses> element within the <ports> element.	The location of a <uses> element is identified or it will go to Step 2.
9	Create a sequence of ConnectionType which contain as many elements as found in Step 8. For at least one <uses> element, create a ConnectionType struct and assign the value of the usesname attribute to the portName field within the portConnectionId field and an object reference to an object implementing an interface not defined in the SCA to the portReference field of the ConnectionType struct.	A ConnectionType sequence of the component uses ports with a port reference implementing an unknown interface is created.
10	Obtain an object reference for a component instance associated with the <componentplacement> element used in Step 2.	An object reference is obtained.
11	Using the object reference from Step 10, invoke the connectUsesPorts with the ConnectionType sequence created in Step 9. (For use-case: 5a)	The connectUsesPorts operation raises an InvalidPort exception.
12	For the same sequence of ConnectionType created in Step 9, replace the portReference field to assign a nil object reference.	A ConnectionType sequence of the component uses ports with a nil port reference is created.

Step	Action	Expected Result
13	Using the object reference from Step 10, invoke the connectUsesPorts with the ConnectionType sequence created in Step 12. (For use-case: 5a)	The connectUsesPorts operation raises an InvalidPort exception.
14	For the same sequence of ConnectionType created in Step 12, replace the portReference field to assign an object reference implementing the interface used by the uses port.	A ConnectionType sequence of the component uses ports with valid port reference is created.
15	Using the object reference from Step 10, invoke the connectUsesPorts with the ConnectionType sequence created in Step 14.	The connectUsesPorts operation doesn't raise an exception.
16	Using the object reference from Step 10, invoke the connectUsesPorts with the ConnectionType sequence created in Step 14 again. (For use-case: 5b)	The connectUsesPorts operation raises an InvalidPort exception.
17	For the same sequence of ConnectionType created in Step 14, replace the connectionId field within the portConnectionId field for all the elements to an arbitrary value (that is not already used) and remove the elements for which the corresponding <uses> elements have a value for the maxconnections attribute equals to 1 or has no value.	A ConnectionType sequence of the component uses ports for which more than one connection can be established.
18	Using the object reference from Step 10, invoke the connectUsesPorts with ConnectionType sequence created in Step 17.	The connectUsesPorts operation doesn't raise an exception.
19	Repeat Steps 17-18 until the connectUsesPorts operation raises an InvalidPort exception. (For use-case: 5d)	The connectUsesPorts operation raise an an InvalidPort exception.
20	Create a sequence of ConnectionType which contain one element that as an arbitrary value for the portName field within the portConnectionId field.	A ConnectionType sequence uses ports unknown to the component.
21	Using the object reference from Step 10, invoke the connectUsesPorts with the ConnectionType sequence created in Step 20. (For use-case: 5c)	The connectUsesPorts operation raise an InvalidPort exception.
22	Go back to Step 2.	N/A

Postconditions: This test procedure modified the operating environment by establishing connections. In order to exercise other runtime tests, the operating environment shall be reset.

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.2.3 Requirement under Test: SCA10

Requirement Text: The *disconnectPorts* operation shall break the connection(s) to the component identified by the input portDisconnections parameter.

Context:

1. The BaseComponent is a component part of an application under test.
2. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
3. The test procedure only tests one component and should be repeated for every component of an application.
4. The SAD file using the component under test is used to determine the connections to be established in the test procedure.

Preconditions:

1. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
2. All connections are expected to be already established.

Test Procedure:

Table 23: Steps to execute Test Procedure for SCA10

Step	Action	Expected Result
1	Demonstrate through some specific actions that the provides port provided to the uses port of the component are used.	Evidence that the provides port are used is demonstrated.
2	Use some testing tools specific ways to disconnect specific connections using the disconnectPorts() operations.	The specified connections are successfully disconnected.
3	Using the same actions in Step 1, demonstrate that the provides port which were provided to the uses port via the connections that have been disconnected in Step 2 of the component are not used.	Evidence that the provides port are disconnected.

Postconditions: N/A

Test Plan Verification Method: Demonstration [Ref1]

Test Plan Result Category: N/A

5.6.2.4 Requirement under Test: SCA11

Requirement Text: The *disconnectPorts* operation shall release all ports if the input portDisconnections parameter is a zero length sequence.

Context:

1. The BaseComponent is a component part of an application under test.
2. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

3. The test procedure only tests one component and should be repeated for every component of an application.
4. The SAD file using the component under test is used to determine the connections to be established in the test procedure.

Preconditions:

1. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
2. All connections are expected to be already established.

Test Procedure:

Table 24: Steps to execute Test Procedure for SCA11

Step	Action	Expected Result
1	Demonstrate through some specific actions that the provides port provided to the uses port of the component are used.	Evidence that the provides port are used is demonstrated.
2	Use some testing tools specific ways to disconnect all connections using the disconnectPorts() operations with a zero length sequence.	The connections are successfully disconnected.
3	Using the same actions in Step 1, demonstrate that all the provides port which were provided to the uses port via the connections that have been disconnected in Step 2 of the component are not used.	Evidence that the provides port are disconnected.

Postconditions: N/A

Test Plan Verification Method: Demonstration [Ref1]

Test Plan Result Category: N/A

5.6.2.5 Requirement under Test: SCA12

Requirement Text: The *disconnectPorts* operation shall raise the InvalidPort exception when the input portDisconnections parameter provides an unknown connection to the *PortAccessor's* component.

Test Plan Objective/Summary: Ensure that an application component, which is a BaseComponent, raises an InvalidPort exception when it is asked to disconnect an unknown connection. The test procedure parses the SCD file to determine the ports, obtain an instance of the component, and invokes the disconnectPorts operation with a connection ID not known by the component.

Context:

1. Context Group 3.2: Common Context related to the validation of an application component.
2. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
3. The ports of a BaseComponent are identified by the <ports> element in the SCD of the component under test.
4. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
5. Section 3.1.3.2.1.2.3.6 of the SCA specification describes four cases where the InvalidPort can be raised. The following cases apply for the disconnectUsesPorts operation:
    a. The connectionId is invalid.
    b. The uses port portName does not exist for the given connectionId.

Preconditions:

1. The existence of a capability on the test platform that is able to establish connections through the PortAccessor interface between a uses port component and provides port component.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
4. The domain profile of the BaseComponent includes an SCD file that contains one or more port definitions.

Test Procedure:

Table 25: Steps to execute Test Procedure for SCA12

Step	Action	Expected Result
1	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
2	Perform steps defined in sub-procedure “Locate next component or assembly”.	See sub-procedure 6.1.
3	Go to step 4 if the element being evaluated is a <componentplacement> or return to step 1 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
4	Open the SPD file indicated by the <componentfile> referenced by the <componentplacement> and locate the <descriptor> element.	The <descriptor> element will be found or the verification will go back to step 2.
5	Determine if the file identified by <localfile> element referenced by the <descriptor> element exists.	The file identified by the <localfile> reference within the <descriptor> subelement exists.
6	Open the SCD file indicated by the <localfile> referenced by the <descriptor> and locate the <componentfeatures> element within the SCD.	The <componentfeatures> element is found within the SCD.

Step	Action	Expected Result
7	Locate the <ports> element within <componentfeatures> element.	The location of the <ports> element is identified.
8	Identify all the <uses> element within the <ports> element.	The location of a <uses> element is identified or it will go to step 2.
9	Create a sequence of ConnectionType which contain as many elements as found in Step 8. For at least one <uses> element, create a ConnectionType struct and assign the value of the usesname attribute to the portName field within the portConnectionId field and an object reference implementing the interface used by the uses port to the portReference field of the ConnectionType struct.	A ConnectionType sequence of the component uses ports with a valid port reference is created.
10	Obtain an object reference for a component instance associated with the <componentplacement> element used in Step 2.	An object reference is obtained.
11	Using the object reference from Step 10, invoke the connectUsesPorts with the ConnectionType sequence created in Step 9.	The connectUsesPorts operation doesn't raises an exception.
12	Create a sequence of ConnectionIdType which contain one element. For that element, create a ConnectionIdType struct and assign an arbitrary value to the connectionId field of the ConnectionIdType struct. (For use-case: 5a and 5b)	A ConnectionIdType sequence with an arbitrary connection id is created.
13	Using the object reference from Step 10, invoke the disconnectPorts with ConnectionIdType sequence created in Step 12.	The disconnectPorts operation raises an InvalidPort exception.
14	Create a sequence of ConnectionIdType which contain an element that is a duplicate of each portConnectionId field of ConnectionType sequence created in Step 9.	A ConnectionIdType sequence is created.
15	Using the object reference from Step 10, invoke the disconnectPorts with ConnectionIdType sequence created in Step 14.	The disconnectPorts operation doesn't raises an exception.
16	Using the object reference from Step 10, invoke the disconnectPorts with ConnectionIdType sequence created in Step 14 again. (For use-case: 5a and 5b)	The disconnectPorts operation raises an InvalidPort exception.
17	Go back to Step 2.	

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.2.6 Requirement under Test: SCA13, SCA14, SCA430

Requirement Text:

SCA13: The *getProvidesPorts* operation shall return the object references that are associated with the input port names and the connectionIds.

SCA13 (from Errata Sheet): The *getProvidesPorts* operation shall return the object references that are associated with the input port names (that are stated in the SCD) within the portConnections parameter.

SCA14: The *getProvidesPorts* operation shall raise an InvalidPort exception when the input portConnections parameter requests undefined connection(s).

SCA430: A BaseComponent shall supply ports for all the ports defined in its domain profile.

Test Plan Objective/Summary: Ensure that an application component, which is a BaseComponent, implements port objects defined in the application component SCD file. The test procedure parses the SCD file to determine the ports, obtain an instance of the component. For the provides ports, the getProvidesPorts operation is invoked to ensure the provides ports are supplied or an exception is raised. For the uses ports, the connectUsesPorts operation is invoked to ensure the uses ports are supported.

Context:

1. The BaseComponent implements the Connectable UoF.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
4. The ports of a BaseComponent are identified by the <ports> element in the SCD of the component under test.
5. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
6. For SCA13, the requirement text from the Errata Sheet has been considered for writing of this procedure.
7. Section 3.1.3.2.1.2.3.6 of the SCA specification describes four cases where the InvalidPort can be raised. The following cases apply for the getProvidesPorts operation:
    a. The provides port portName does not exist.

Preconditions:

1. The existence of a capability on the test platform that is able to establish connections through the PortAccessor interface between a uses port component and provides port component.

2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
4. The domain profile of the BaseComponent includes an SCD file that contains one or more port definitions.

Test Procedure:

Table 26: Steps to execute Test Procedure for SCA13, SCA14, SCA430

Step	Action	Expected Result
1	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
2	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
3	Go to step 4 if the element being evaluated is a <componentplacement> or return to step 1 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
4	Open the SPD file indicated by the <componentfile> referenced by the <componentplacement> and locate the <descriptor> element.	The <descriptor> element will be found or the verification will go back to step 2.
5	Determine if the file identified by <localfile> element referenced by the <descriptor> element exists.	The file identified by the <localfile> reference within the <descriptor> subelement exists.
6	Open the SCD file indicated by the <localfile> referenced by the <descriptor> and locate the <componentfeatures> element within the SCD.	The <componentfeatures> element is found within the SCD.
7	Locate the <ports> element within <componentfeatures> element.	The location of the <ports> element is identified.
8	Identify all the <provides> element within the <ports> element.	The location of a <provides> element is identified or it will go to step 16.
9	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
10	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2.
11	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
12	Obtain the providesPorts field of the ComponentType. (SCA13)	The list of ports provided by the component at registration is obtained.

Step	Action	Expected Result
13	Create a list of ports name containing all the provides ports supported by the component as obtained in Step 8 excluding the port names listed in the providesPorts field obtained in Step 12. (SCA13)	A list of provides ports not provided at component registration.
14	Create a sequence of ConnectionType which contain as many elements as found in Step 13. For each <provides> element, create a ConnectionType struct and assign the value of the providesname attribute to the portName field within the portConnectionId field of the ConnectionType struct. (SCA13)	A ConnectionType sequence of the component provides ports not provided during registration is created.
15	Using the object reference from Step 11, invoke the getProvidesPorts with ConnectionType sequence created in Step 14. (SCA13)	The getProvidesPorts operation returns without an exception.
16	Create a sequence of ConnectionType which contain one element that as an arbitrary value for the portName field within the portConnectionId field. (SCA14)	A ConnectionType sequence provides ports unknown to the component.
17	Using the object reference from Step 11, invoke the getProvidesPorts with ConnectionType sequence created in Step 16. (SCA14)	The getProvidesPorts operation raises an invalid port exception.
18	Identify all the <uses> elements within the <ports> element. (SCA430)	The location of a <uses> element is identified or it will go to step 2.
19	Create a sequence of ConnectionType which contain as many elements as found in Step 18. For each <uses> element, create a ConnectionType struct and assign the value of the usesname attribute to the portName field within the portConnectionId field and a non-nil value to the portReference field of the ConnectionType struct. (SCA430)	A ConnectionType sequence of the component uses ports is created.
20	Using the object reference from Step 11, invoke the connectUsesPorts with ConnectionType sequence created in Step 19. (SCA430)	The connectUsesPorts operation returns without an exception.
21	Go back to Step 2.	N/A

Postconditions: This test procedure modified the operating environment by establishing connections. In order to exercise other runtime tests, the operating environment shall be reset.

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.2.7 Requirement under Test: SCA519

Requirement Text: The *connectUsesPorts* operation shall disconnect any connections it formed if any connections in the input portConnections parameter cannot be successfully established.

Test Plan Objective/Summary: Ensure that an application component, which is a BaseComponent, disconnects any connections if one of the specified connections is invalid. The test procedure parses the SCD file to determine the ports, obtain an instance of the component, and check the compliance to the requirement by invoking the disconnectPorts operation on connections which shall not be connected.

Context:

1. Context Group 3.2: Common Context related to the validation of an application component.
2. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
3. The ports of a BaseComponent are identified by the <ports> element in the SCD of the component under test.
4. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. The existence of a capability on the test platform that is able to establish connections through the PortAccessor interface between a uses port component and provides port component.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
4. The domain profile of the BaseComponent includes an SCD file that contains one or more port definitions.
5. This test procedure requires that the requirement SCA12 should be verified.

Test Procedure:

Table 27: Steps to execute Test Procedure for SCA519

Step	Action	Expected Result
1	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
2	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
3	Go to step 4 if the element being evaluated is a <componentplacement> or return to step 1 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification will terminate.
4	Open the SPD file indicated by the <componentfile> referenced by the <componentplacement> and locate the <descriptor> element.	The <descriptor> element is found or the verification will go back to step 2.
5	Determine if the file identified by <localfile> element referenced by the <descriptor> element exists.	The file identified by the <localfile> reference within the <descriptor> subelement exists.
6	Open the SCD file indicated by the <localfile> referenced by the <descriptor> and locate the <componentfeatures> element within the SCD.	The <componentfeatures> element is found within the SCD.
7	Locate the <ports> element within <componentfeatures> element.	The location of the <ports> element is identified.
8	Identify all the <uses> element within the <ports> element.	The location of a <uses> element is identified or it will go to Step 2.
9	Create a sequence of ConnectionType which contain as many elements as found in Step 8. For each <uses> element, create a ConnectionType struct and assign the value of the usesname attribute to the portName field within the portConnectionId field and an object reference implementing the interface used by the uses port.	A ConnectionType sequence of the component uses ports with valid port reference is created.
10	Obtain an object reference for a component instance associated with the <componentplacement> element used in Step 2.	An object reference is obtained.
11	Using the object reference from Step 10, invoke the connectUsesPorts with the ConnectionType sequence created in Step 9.	The connectUsesPorts operation doesn't raise an exception.
12	Create a sequence of ConnectionIdType which contain as many elements as the sequence of ConnectionType created in step 9. For each element, duplicate the ConnectionIdType struct of each element of the sequence of ConnectionType created in step 9.	A ConnectionIdType sequence is created.

Step	Action	Expected Result
13	Using the object reference from Step 10, invoke the disconnectPorts with the ConnectionIdType sequence created in Step 12.	The disconnectPorts operation doesn't raise an exception.
14	For the same sequence of ConnectionType created in Step 9, add one the ConnectionType elements with an arbitrary value for the portName field within the portConnectionId field to the end of the sequence.	A ConnectionType sequence of the component uses ports with one invalid ConnectionType element.
15	Using the object reference from Step 10, invoke the connectUsesPorts with the ConnectionType sequence created in Step 14.	The connectUsesPorts operation raises an InvalidPort exception when using the invalid element.
16	Using the sequence of ConnectionIdType created in Step 12, create as many sequences of ConnectionIdType of length 1, each containing one of the elements contained in the original sequence.	As many ConnectionIdType sequence as the length of the ConnectionIdType sequence created in step 12 are created.
17	Using the object reference from Step 10, invoke the disconnectPorts operation with each sequence created in Step 16.	The disconnectPorts operation raises an InvalidPort exception for each sequence.
18	Using the ConnectionType sequence created in step 14, switch the first and the last element of the sequence to have the invalid ConnectionType element at the beginning of the sequence and repeat steps 15 to 17.	Expected result is the same at all steps. The goal is to test the use case where the invalid connection is the first element of the sequence.
19	Go back to Step 2.	N/A

Postconditions: This test procedure modified the operating environment by establishing connections. In order to exercise other runtime tests, the operating environment shall be reset.

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.2.8 Requirement under Test: SCA547

Requirement Text: A BaseComponent shall realize the PortAccessor interface as a proxy for its uses and provides ports.

Test Plan Objective/Summary: Ensure an ApplicationComponent, which is a BaseComponent, inherits the PortAccessor interface as per realization of the PortAccessor IDL interface. The test procedure obtains an instance of the component and narrows it to CF::PortAccessor to validate the requirement.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 28: Steps to execute Test Procedure for SCA547

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the PortAccessor interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the PortAccessor interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Determine that the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contain a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.
9	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::PortAccessor interface.	The componentObject can be narrowed to CF::PortAccessor interface.
11	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.3 LifeCycle UoF

5.6.3.1 Requirement under Test: SCA15

Requirement Text: The *initialize* operation shall raise an InitializeError exception when an initialization error occurs.

Test Plan Objective/Summary: Locate the source code of the BaseComponent and search for the initialize operation to ensure an InitializeError exception is raised when an error occurs.

Context:

1. Context Group 3.4: Common Context for requirements associated with BaseComponent (not including a sub-application).

Preconditions:

1. The application developer provides the source file name for the BaseComponent as identified by item 2 in Context Group 3.4 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the BaseComponent.
2. The SCD of the component under test indicates that it supports the LifeCycle interface.

Test Procedure:

Table 29: Steps to execute Test Procedure for SCA15

Step	Action	Expected Result
1	Perform steps defined in sub-procedure “Ensure an operation raises a specific exception” with ‘initialize()’ as SCA operation input argument and ‘InitializeError’ as exception input argument.	See sub-procedure 6.6.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Result Category: N/A

5.6.3.2 Requirement under Test: SCA432

Requirement Text: A BaseComponent shall realize the *LifeCycle* interface.

Test Plan Objective/Summary: Ensure a BaseComponent inherits the LifeCycle interface as per realization of the LifeCycle IDL interface. The test procedures obtain an instance of the component and narrow it to CF::LifeCycle to validate the requirement.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).

2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 30: Steps to execute Test Procedure for SCA432

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the LifeCycle interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the LifeCycle interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Determine that the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contain a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.
9	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::LifeCycle interface.	The componentObject can be narrowed to CF::LifeCycle interface.
11	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.4 Releaseable UoF

5.6.4.1 Requirement under Test: SCA16

Requirement Text: The *releaseObject* operation shall release all internal memory allocated by the component during the life of the component.

No test is provided to address this requirement.

The intent of this requirement is to enhance the robustness and reliability of SCA components by ensuring that they have a comprehensive approach to memory management that guards against memory leaks that might compromise platform reliability. This functionality is critical to platform operations and is a universally accepted goal of sound programming practices. The SCA requirement reinforces that this management should be performed, but in truth, the implementation of this principle exceeds the SCA's area of responsibility. A complete memory management policy is not only applicable to code developed in support of SCA operations, but also for the code that implements the application business logic, and any libraries or support artifacts included with the deployed product. We believe that the level of effort required to comprehensively test this requirement exceeds the reasonable bounds, and capabilities, of what can be done within the scope of compliance verification.

Several approaches could be implemented to perform partial validation of this requirement, but we believe that pursuing this approach would lead to a false sense of security. Therefore, we recommend that this requirement is not assessed as part of the SCA compliance verification process.

Our position does not advocate ignoring this objective, we believe that it should be monitored throughout the software development process and assessed at some point within the development cycle or as part of final qualification or acceptance testing when the product developer has access to tools that can analyze the component artifacts. Example instrumentation tools like Memcheck by Valgrind, Electric Fence, etc can help perform dynamic debugging. Another approach would be to use static analysis tools like HP Fortify, Klocwork, Coverity, etc which evaluate the code to ensure proper usage memory allocation and deallocation commands. Ideally, both techniques can be used in conjunction to evaluate all of the component artifacts.
Since proper memory management is so important, we also recommend that a customer or acquiring organization require proof of fulfillment of this functionality as part of their acceptance process. The proof could be artifacts generated by the analysis tools or successful completion of other product specific tests that measure and report memory utilization and availability.

5.6.4.2 Requirement under Test: SCA17

Requirement Text: The *releaseObject* operation shall tear down the component and release it from the operating environment.

Test Plan Objective/Summary: Ensure a component is no longer usable after it has been released. The test procedure obtains an instance of the component before it is released and try to invoke an operation after the component is released.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 31: Steps to execute Test Procedure for SCA17

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the LifeCycle interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the LifeCycle interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2. The ComponentType of the ApplicationComponent is obtained.
9	Obtain the componentObject field of the ComponentType of the application component corresponding to the <componentinstantiation> element ID.	The componentObject is found and the result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::LifeCycle interface.	The componentObject can be narrowed to CF::LifeCycle interface.
11	Invoke the releaseObject() operation on the narrowed componentObject.	The application component is released.
12	Invoke the releaseObject() operation again.	CORBA::object_not_exist exception is found.
13	Repeat steps 7-12 until no more <componentinstantiation> elements are found within the <componentplacement> element, otherwise go to step 3.	The next componentinstantiation will be evaluated or the verification will go to step 3.

Postconditions: The test environment needs to be reset because the Application Manager is still alive.

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.4.3 Requirement under Test: SCA18

Requirement Text: The *releaseObject* operation shall raise a ReleaseError exception when a release error occurs.

Test Plan Objective/Summary: Locate the source code of the BaseComponent and search for the releaseObject operation to ensure a ReleaseError exception is raised when an error occurs.

Context:

1. Context Group 3.4: Common Context for requirements associated with BaseComponent (not including a sub-application).

Preconditions:

1. The application developer provides the source file name for the BaseComponent as identified by item 2 in Context Group 3.4 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the BaseComponent.
2. The SCD of the component under test indicates that it supports the LifeCycle interface.

Test Procedure:

Table 32: Steps to execute Test Procedure for SCA18

Step	Action	Expected Result
1	Perform steps defined in sub-procedure “Ensure an operation raises a specific exception” with ‘releaseObject()’ as SCA operation input argument and ‘ReleaseError’ as exception input argument.	See sub-procedure 6.6.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Result Category: N/A

5.6.4.4 Requirement under Test: SCA518

Requirement Text: The *releaseObject* operation shall raise a ReleaseError exception when a release error occurs.

Test Plan Objective/Summary: Ensure that any remaining connection are disconnected when an application component is released and that the object reference that were provided with the connections are no longer used.

Context:

1. Context Group 3.4: Common Context for requirements associated with BaseComponent (not including a sub-application).

Preconditions:

1. The application developer provides the source file name for the BaseComponent as identified by item 2 in Context Group 3.4 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the BaseComponent.
2. The SCD of the component under test indicates that it supports the LifeCycle interface.

Test Procedure:

Table 33: Steps to execute Test Procedure for SCA518

Step	Action	Expected Result
1	Determine the source file names associated with the SPD for the BaseComponent (see Preconditions for this information).	The source file names associated with the SPD for the BaseComponent is determined.
2	Search the source files of the BaseComponent for the releaseObject operation and ensure that it invokes the disconnectPorts SCA operation or any other internal port disconnection operation for any remaining connections and ensure that the internal association between the connectionId/portName pair (defined as ConnectionIdType) and the object reference provided when a connection was established is removed.	The releaseObject operation is found and it calls the disconnectPorts operation or the internal association between the connectionId/portName pair (defined as ConnectionIdType) and the object reference provided when a connection was established is removed.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Result Category: N/A

5.6.5 Configurable UoF

5.6.5.1 Requirement under Test: SCA26, SCA27, SCA28

Requirement Text:

SCA26: The *configure* operation shall assign values to the properties as indicated in the input configProperties parameter.

SCA27: The configure operation shall raise a PartialConfiguration exception when some configuration properties were successfully set and some configuration properties were not successfully set.

SCA28: The configure operation shall raise an InvalidConfiguration exception when a configuration error occurs and no configuration properties were successfully set.

Test Plan Objective/Summary: Ensure an ApplicationComponent, which is a BaseComponent, only assigns values for the specific configurable properties (i.e. defined within the component's PRF files) that are provided as input parameters to the configure operation. The test procedure extracts the set of configure properties defined within the profile, issues the configure command and then queries the queryable properties and compares the results. Then the test procedure repeats the same process twice, the first time with some valid and some invalid properties, the second with all invalid properties.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseFactoryComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 34: Steps to execute Test Procedure for SCA26, SCA27, SCA28

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.

Step	Action	Expected Result
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the PropertySet interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the PropertySet interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Perform steps in sub-procedure “Find Component Type of a component instance”.	See sub-procedure 6.2. The ComponentType of the ApplicationComponent is obtained.
9	Obtain the implementation id in the specializedInfo field of the ComponentType of the application component corresponding to the <componentinstantiation> element ID.	The implementation id of the component instance is obtained.
10	Perform steps defined in sub-procedure “Obtain component configure properties” with implementation id obtained in step 9 as component implementation id input argument and with ‘writeonly and readwrite’ as the property modes input argument.	See sub-procedure 6.7.
11	Obtain the componentObject field of the ComponentType of the application component corresponding to the <componentinstantiation> element ID.	The componentObject is found and the result of the CORBA::is_nil operation on componentObject field is false.
12	Narrow the componentObject to the CF::PropertySet interface.	The componentObject can be narrowed to CF::PropertySet interface.
13	Construct a list of all the component’s configurable properties with valid values to be used as an input parameter.	The configure command input parameter list has been built.
14	Invoke the configure() operation on the narrowed componentObject with the input parameter constructed in the previous step. (SCA26)	The component’s configure operation is successfully invoked.
15	Invoke the query() operation on the narrowed componentObject with an input parameter of zero size. (SCA26)	The component’s queryable properties are returned in the operation’s input parameter.
16	Compare the values passed to the configure operation against those returned with the corresponding queryable properties. (SCA26)	The assigned values are successfully compared against the queryable properties.
17	Replace the values of a random number, x < n, of the previously identified configurable property IDs with invalid values when there is more than 1 configure property or go to step 21. (SCA27)	The configure command input parameter list has been built with invalid property value(s) or go to step 21.

Step	Action	Expected Result
18	Invoke the configure() operation on the narrowed componentObject with the input parameter constructed in the previous step. (SCA27)	The component's configure operation returns a PartialConfiguration exception.
19	Invoke the query() operation on the narrowed componentObject with an input parameter of zero size. (SCA27)	The component's queryable properties are returned in the operation's input parameter.
20	Compare the values passed to the configure operation against those returned with the corresponding queryable properties. (SCA27)	The assigned values of the valid properties are successfully compared against the queryable properties and the values for those properties listed in the PartialConfiguration exception have not changed.
21	Perform steps defined in sub-procedure "Obtain component configure properties" with implementation id obtained in step 9 as component implementation id input argument and with 'readonly' as the property modes input argument. (SCA27)	See sub-procedure 6.7.
22	If the component has some properties with 'readonly' mode, construct a list of all the component's properties obtained in Step 21 to be used as an input parameter or go to step 24. (SCA27)	The configure command input parameter list has been built with 'readonly'invalid properties or go to step 24.
23	Invoke the configure() operation on the narrowed componentObject with the input parameter constructed in the previous step. (SCA27)	The component's configure() operation returns a PartialConfiguration exception.
24	Construct a list of a random number of the component's configurable properties, all with invalid values to be used as an input parameter. (SCA28)	The configure() command input parameter list has been built.
25	Invoke the configure() operation on the narrowed componentObject with the input parameter constructed in the previous step.(SCA28)	The component's configure() operation returns an InvalidConfiguration exception.
26	Invoke the query() operation on the narrowed componentObject with an input parameter of zero size. (SCA28)	The component's queryable properties are returned in the operation's input parameter.
27	Compare the values passed to the configure operation against those returned with the corresponding queryable properties. (SCA28)	The assigned values for those properties listed in the InvalidConfiguration exception have not changed.
28	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.5.2 Requirement under Test: SCA29, SCA30, SCA31

Requirement Text:

SCA29: The query operation shall return all component properties when the inout parameter configProperties is zero size

SCA30: The query operation shall return only those id/value pairs specified in the configProperties parameter if the parameter is not zero size.

SCA31: The query operation shall raise the CF::UnknownProperties exception when one or more properties being requested are not known by the component.

Test Plan Objective/Summary: Ensure an ApplicationComponent, which is a BaseComponent, only returns values for the specific queryable properties (i.e. defined within the component's PRF files) that are provided as input parameters to the query operation or for all properties if no property is provided as input. The test procedure extracts the set of query properties defined within the profile, selects a subset of the properties, issues the query command and compares the properties returned against those that were provided or defined. Then the test procedure invokes the query command once more with an invalid property.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseFactoryComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 35: Steps to execute Test Procedure for SCA29, SCA30, SCA31

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.

Step	Action	Expected Result
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure “Locate next component or assembly”.	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure “Obtain supported interface list”	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the PropertySet interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the PropertySet interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Perform steps in sub-procedure “Find Component Type of a component instance”.	See sub-procedure 6.2. The ComponentType of the ApplicationComponent is obtained.
9	Obtain the implementation id in the specializedInfo field of the ComponentType of the application component corresponding to the <componentinstantiation> element ID.	The implementation id of the component instance is obtained.
10	Perform steps defined in ““Obtain component configure properties” with implementation id obtained in step 9 as component implementation id input argument and with ‘readonly and readwrite’ as the property modes input argument.	See sub-procedure 6.7
11	Obtain the componentObject field of the ComponentType of the application component corresponding to the <componentinstantiation> element ID.	The componentObject is found and the result of the CORBA::is_nil operation on componentObject field is false.
12	Narrow the componentObject to the CF::PropertySet interface.	The componentObject can be narrowed to CF::PropertySet interface.
13	Invoke the query() operation on the narrowed componentObject with an input parameter of zero size. (SCA29)	The component’s queryable properties are returned in the operation’s input parameter.
14	Compare the returned properties against those obtained from the component domain profile files. (SCA29)	All the queryable properties from the domain profile are present in the returned value.

Step	Action	Expected Result
15	Construct a random sized list of at most n-1 of the component's queryable properties or a list of 1 if there is only 1 property to be used as an input parameter. (SCA30 and SCA31)	The query command input parameter list has been built.
16	Invoke the query() operation on the narrowed componentObject with the input parameter constructed in the previous step. (SCA30)	All properties are returned with values in the operation's input parameter.
17	Append the word "Invalid" to a random number, n >= 1, of the previously identified queryable property IDs and clear any value from the data structure. (SCA31)	The query command input parameter list has been built with invalid property(ies).
18	Invoke the query() operation on the narrowed componentObject with the input parameter constructed in the previous step. (SCA31)	The query operation will raise the UnknownProperties exception.
19	Perform steps defined in sub-procedure "Obtain component configure properties" with implementation id obtained in step 9 as component implementation id input argument and with 'writeonly' as the property modes input argument. (SCA31)	See sub-procedure 6.7
20	If the component has some properties with 'writeonly' mode, construct a list of all the component's properties obtained in Step 19 to be used as an input parameter or go to step 22. (SCA31)	The query command input parameter list has been built with 'writeonly' invalid properties or go to step 20.
21	Invoke the query() operation on the narrowed componentObject with the input parameter constructed in the previous step. (SCA31)	The component's query() operation returns an UnknownProperties exception.
22	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.5.3 Requirement under Test: SCA429

Requirement Text: A BaseComponent shall configure or retrieve query values for all properties whose kindtype is "configure" as defined in its domain profile.

No test is provided since the test procedures for SCA26, SCA29 and SCA30 address this requirement.

5.6.5.4 Requirement under Test: SCA545

Requirement Text: A BaseComponent shall realize the PropertySet interface to configure and query its properties.

Test Plan Objective/Summary: Ensure an ApplicationComponent, which is a BaseComponent, inherits the PropertySet interface as per realization of the PropertySet IDL interface. The test procedure obtains an instance of the component and narrows it to CF::PropertySet to validate the requirement.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 36: Steps to execute Test Procedure for SCA545

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the PropertySet interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the PropertySet interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Determine that the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contain a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.
9	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::PropertySet interface.	The componentObject can be narrowed to CF::PropertySet interface.
11	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.6 Controllable UoF

5.6.6.1 Requirement under Test: SCA34

Requirement Text: The *start* operation shall raise the StartError exception if an error occurs while starting the component.

Test Plan Objective/Summary: Locate the source code of the BaseComponent and search for the start operation to ensure a StartError exception is raised when an error occurs.

Context:

1. Context Group 3.4: Common Context for requirements associated with BaseComponent (not including a sub-application).

Preconditions:

1. The application developer provides the source file name for the BaseComponent as identified by item 2 in Context Group 3.4 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the BaseComponent.
2. The SCD of the component under test indicates that it supports the Controllable interface.

Test Procedure:

Table 37: Steps to execute Test Procedure for SCA34

Step	Action	Expected Result
1	Perform steps defined in sub-procedure “Ensure an operation raises a specific exception” with ‘start()’ as SCA operation input argument and ‘StartError as exception input argument.	See sub-procedure 6.6.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Result Category: N/A

5.6.6.2 Requirement under Test: SCA37

Requirement Text: The *stop* operation shall raise the StopError exception if an error occurs while stopping the component.

Test Plan Objective/Summary: Locate the source code of the BaseComponent and search for the start operation to ensure a StopError exception is raised when an error occurs.

Context:

1. Context Group 3.4: Common Context for requirements associated with BaseComponent (not including a sub-application).

Preconditions:

1. The application developer provides the source file name for the BaseComponent as identified by item 2 in Context Group 3.4 to the verifier. This is an unnecessary precondition if the verifier is able to determine the source file of the BaseComponent.
2. The SCD of the component under test indicates that it supports the Controllable interface.

Test Procedure:

Table 38: Steps to execute Test Procedure for SCA34

Step	Action	Expected Result
1	Perform steps defined in sub-procedure "Ensure an operation raises a specific exception" with 'stop()' as SCA operation input argument and 'StopError as exception input argument.	See sub-procedure 6.6.

Postconditions: N/A

Test Plan Verification Method: Inspection [Ref1]

Test Plan Result Category: N/A

5.6.6.3 Requirement under Test: SCA433, SCA32, SCA33, SCA36

Requirement Text:

SCA433: A BaseComponent shall realize the *ControllableInterface* interface to provide overall management control of the component.

SCA32: The readonly started attribute shall return the component's started value.

SCA33: The *start* operation shall set the started attribute to a value of TRUE.

SCA36: The *stop* operation shall set the started attribute to a value of FALSE.

Test Plan Objective/Summary: Ensure an ApplicationComponent, which is a BaseComponent, inherits the ControllableInterface interface as per realization of the ControllableInterface IDL interface. The test procedure obtains an instance of the component and narrows it to that interface to validate the requirement (SCA433). The test procedure also ensures that the started attribute of the ControllableInterface interface returns the appropriate value after the start and stop operations have been successfully invoked (SCA32, SCA33, SCA36).

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 39: Steps to execute Test Procedure for SCA433, SCA32, SCA33, SCA36

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ApplicationManager reference is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the ControllableInterface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the ControllableInterface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Determine that the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contain a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.

Step	Action	Expected Result
9	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::ControllableInterface interface. (SCA433)	The componentObject can be narrowed to CF::ControllableInterface interface.
11	Invoke the start() operation on the narrowed componentObject. (SCA32)(SCA33)	The start() operation is invoked successfully.
12	Retrieve the value of the started attribute from the CF::ControllableInterface interface. (SCA32)(SCA33)	The value of the started attribute is TRUE.
13	Invoke the start() operation on the narrowed componentObject again. (SCA32)(SCA33)	The application component does not raise an exception.
14	Retrieve the value of the started attribute from the CF::ControllableInterface interface. (SCA32)(SCA33)	The value of the started attribute is TRUE.
15	Invoke the stop() operation on the narrowed componentObject. (SCA36)	The stop() operation is invoked successfully.
16	Retrieve the value of the started attribute from the CF::ControllableInterface interface. (SCA32)(SCA36)	The value of the started attribute is FALSE.
17	Invoke the stop() operation on the narrowed componentObject again. (SCA36)	The application component does not raise an exception.
18	Retrieve the value of the started attribute from the CF::ControllableInterface interface. (SCA32)(SCA36)	The value of the started attribute is FALSE.
19	Invoke the start() operation on the narrowed componentObject. (SCA32)(SCA33)	The start() operation is invoked successfully.
20	Retrieve the value of the started attribute from the CF::ControllableInterface interface. (SCA32)(SCA33)	The value of the started attribute is TRUE.
21	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.7 LogProducer and Configurable UoFs

5.6.7.1 Requirement under Test: SCA420

Requirement Text: A BaseComponent shall implement a 'configure' kind of property with a name of PRODUCER_LOG_LEVEL.

Test Plan Objective/Summary: Ensure that an application component, which is a BaseComponent, that implements the PropertySet interface supports a property named PRODUCER_LOG_LEVEL. Support is determined by calling the configure and query operations.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 has been verified.

Test Procedure:

Table 40: Steps to execute Test Procedure for SCA420

Step	Action	Expected Result
1	Obtain the ComponentType struct of the ApplicationManager instance of the application under test.	The ComponentType struct is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the PropertySet interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the PropertySet interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2.
9	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::PropertySet interface.	The componentObject can be narrowed to CF::PropertySet interface.
11	Invoke the configure operation with the argument of a property with an id of PRODUCER_LOG_LEVEL and a value of a valid log level.	The configure operation returns without an exception.
12	Invoke the query operation with the argument of a property with an id of PRODUCER_LOG_LEVEL.	The query operation returns without an exception and the value is the same as provided in step 11.
13	Return to step 3	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.8 Log Producer UoF

5.6.8.1 Requirement under Test: SCA421

Requirement Text: A BaseComponent shall output only those log records to a log service that correspond to enabled log level values in the PRODUCER_LOG_LEVEL attribute.

Test Plan Objective/Summary: Ensure that the ApplicationComponents, which are BaseComponents, generate log messages in accordance with the levels identified by the value of the PRODUCER_LOG_LEVEL configure property. Demonstrate that log messages are produced by the application in accordance with the component's enabled log level.

Context:

1. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
2. A BaseComponent is a component part of an application under test.
3. The log service is configured to support the number of messages logged by the application's components.

Preconditions: N/A

Test Procedure:

Table 41: Demonstration steps for SCA421

Step	Action	Expected Result
1	Empty the system log file(s) and display their contents.	The log file(s) are empty.
2	Enable the valid log levels for each log producing component within the application via the PRODUCER_LOG_LEVEL attribute.	All valid log levels are enabled for each of the application's components.
3	Execute a series of operations that will cause each log producing ApplicationComponent to generate log messages (the desired scenario is to select a series of operations that will result in the application's end state being identical to the start state so they can be executed again and produce the same messages).	The log file(s) are populated with messages generated by each component.
4	Disable at least one of the valid log levels for each component that generates log messages stored within the log file(s).	A subset of log levels for each component are enabled.
5	Execute the same set of operations as performed previously in step 3.	The log file(s) will be appended with messages generated only for the enabled log levels.
6	Repeat steps 4-5 with a different combination of enabled log levels (the execution of steps 4-6 should demonstrate that all of the application components are able to constrain their generated log messages).	The log file(s) will be appended with messages generated only for the enabled log levels.
7	Enable the valid log levels for each log producing component within the application via the PRODUCER_LOG_LEVEL attribute.	All valid log levels are enabled for each of the application's components.
8	Execute the same set of operations as performed previously in step 3.	The log file(s) will be appended with the same messages as those generated in step 3.

Postconditions: N/A

Test Plan Verification Method: Demonstration [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.8.2 Requirement under Test: SCA423

Requirement Text: A BaseComponent shall operate normally in the case where the connections to a log service are nil or an invalid reference.

Test Plan Objective/Summary: Ensure that the ApplicationComponents, which are BaseComponents, operate normally even if they are not connected to a Log service or the reference to the Log service they are using becomes invalid. Demonstrate that the components of the application continue to perform their main work after the connections to the log service becomes invalid.

Context:

1. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
2. A BaseComponent is a component part of an application under test.
3. The log service is configured to support the number of messages logged by the application's components.

Preconditions: N/A

Test Procedure:

Table 42: Demonstration steps for SCA423

Step	Action	Expected Result
1	Empty the system log file(s) and display their contents.	The log file(s) are empty.
2	Enable the valid log levels for each log producing component within the application via the PRODUCER_LOG_LEVEL attribute.	All valid log levels are enabled for each of the application's components.
3	Execute a series of operations that will cause each log producing ApplicationComponent to generate log messages (the desired scenario is to select a series of operations that will result in the application's end state being identical to the start state so they can be executed again and produce the same messages).	The log file(s) are populated with messages generated by each component.
4	Use some testing tools specific ways to disconnect all connections to the Log service.	Connections to Log service are successfully disconnected.
5	Execute the same set of operations as performed previously in step 3.	The log file(s) are not populated with messages generated by each component and each component behaves properly to perform their main work.
6	Use some testing tools specific ways to connect the application components that are LogProducer to the Log service.	Connections to Log service are successfully established.
7	Execute the same set of operations as performed previously in step 3.	The log file(s) will be appended with the same messages as those generated in step 3.
8	Use some platforms and/or testing tools specific ways to invalidate the reference to the Log service used by the application components.	The reference to the Log service used by the application components is invalid.
9	Execute the same set of operations as performed previously in step 3.	The log file(s) are not populated with messages generated by each component and each component behaves properly to perform their main work.
10	Use some platforms and/or testing tools specific ways to make valid the reference to the Log service used by the application components.	The reference to the Log service used by the application components is valid.
11	Execute the same set of operations as performed previously in step 3.	The log file(s) are not populated with messages generated by each component and each component behaves properly to perform their main work.

Postconditions: N/A

Test Plan Verification Method: Demonstration [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.9 Event Producer UoF

5.6.9.1 Requirement under Test: SCA424

Requirement Text: A BaseComponent that produces events shall implement the CosEventComm::PushSupplier interface and use the CosEventComm::PushConsumer interface for generating the events.

Test Plan Objective/Summary: Ensure that an application component that produces events implements the OMG Event Service Specification CosEventComm::PushSupplier interface and uses the CosEventComm::PushConsumer interface for generating events.

Context:

1. The CosEventComm module is used by consumers for receiving events and by producers for generating events.
2. The CosEventComm interfaces are specified in the OMG Event Service Specification v1.2 (October 2004). https://www.omg.org/spec/EVNT/About-EVNT/
3. It is assumed that the interfaces are implemented via inheritance and it is this inheritance relationship that identifies application components or ports to which this requirement is applicable.
4. An application component that produces events should inherit from CosEventComm::PushSupplier and CosEventComm::PushConsumer defined in the OMG Event Service Specification under the CosEventComm Module.
5. In general, it is not possible, except through deep manual code inspection, to determine that an application component is not using some other *ad hoc* event interface.
6. Each source file that implements an application component class should be checked for an inheritance relationship with an interface specified in the OMG Event Service Specification.
7. Each source file that declares or defines an application component which inherits (directly or transitively) from an OMG Event Service Specification interface must be analyzed.
8. The requirement is not applicable to virtual classes. A virtual class is not realizable and therefore cannot produce or generate events.
9. Each source file must be analyzed in the context of full execution of all preprocessor directives.
    - 9.1 All preprocessor-relevant compiler flags are specified
    - 9.2 All included files are available
    - 9.3 All conditional directives are executed and excluded code is eliminated
10. This test does not check for parity between event suppliers and event consumers.

Preconditions:

1. The developer provides the relevant source code files declaring and defining application components.
2. For each source code file, the developer provides all preprocessor-relevant compiler flags and included (including transitively included) files.

Test Procedure:

Table 43: Steps to execute Test Procedure for SCA424

Step	Action	Expected Result
1	Identify source files in scope of SCA424 requirement test.	The source files are identified.
2	Apply preprocessor execution to source files identified in Step 1.	The preprocessor execution directives are applied.
3	For each non-virtual class that is used to implement the BaseComponent, identify those that inherit from CosEventComm::PushSupplier.	The classes that inherit from CosEventComm::PushSupplier are identified.
4	For each class identified in Step 3, verify that the class implements PushSupplier::disconnect_push_supplier().	The class implements the PushSupplier::disconnect_push_supplier() method.
5	Verify that the set of classes identified in Step 3 is non-empty.	The set of classes identified in Step 3 is non-empty.
6	For each class that is used to implement the BaseComponent identify all uses of: PushConsumer::push() and PushConsumer::disconnect_push_consumer().	The classes that use the PushConsumer::push() and PushConsumer::disconnect_push_consumer() method are identified.

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Necessary [Ref1]

5.6.9.2 Requirement under Test: SCA425

Requirement Text: A producer BaseComponent shall not forward or raise any exceptions when the connection to a *CosEventComm*::*PushConsumer* is a nil or invalid reference.

Test Plan Objective/Summary: Ensure that an application component that produces events using the CosEventComm::PushConsumer interface doesn't forward or raise exceptions when the connection provides a nil or invalid object reference.

Context:

1. The CosEventComm module is used by consumers for receiving events and by producers for generating events.
2. The CosEventComm interfaces are specified in the OMG Event Service Specification v1.2 (October 2004). https://www.omg.org/spec/EVNT/About-EVNT/

3. In general, it is not possible, except through deep manual code inspection, to determine that an application component is not using some other *ad hoc* event interface.
4. Each source file that implements an application component class should be checked for an inheritance relationship with an interface specified in the OMG Event Service Specification.
5. Each source file that declares or defines an application component which inherits (directly or transitively) from an OMG Event Service Specification interface must be analyzed.
6. The requirement is not applicable to virtual classes. A virtual class is not realizable and therefore cannot produce or generate events.
7. Each source file must be analyzed in the context of full execution of all preprocessor directives.
    7.1 All preprocessor-relevant compiler flags are specified
    7.2 All included files are available
    7.3 All conditional directives are executed and excluded code is eliminated
8. This test does not check for parity between event suppliers and event consumers.

Preconditions:

1. The developer provides the relevant source code files declaring and defining application components.
2. For each source code file, the developer provides all preprocessor-relevant compiler flags and included (including transitively included) files.

Test Procedure:

Table 44: Steps to execute Test Procedure for SCA425

Step	Action	Expected Result
1	Identify source files in scope of SCA425 requirement test.	The source files are identified.
2	Apply preprocessor execution to source files identified in Step 1.	The preprocessor execution directives are applied.
3	For each non-virtual class that is used to implement the BaseComponent, identify those that inherit from CosEventComm::PushSupplier.	The classes that inherit from CosEventComm::PushSupplier are identified.
4	Verify that the set of classes identified in Step 3 is non-empty.	The set of classes identified in Step 3 is non-empty.
5	For each class that is used to implement the BaseComponent identify all uses of: PushConsumer::push() and PushConsumer::disconnect_push_consumer().	The classes that use the PushConsumer::push() and PushConsumer::disconnect_push_consumer() method are identified.
6	For each use of PushConsumer::push() identified in step 5 ensure that exceptions are not forwarded or raised if the consumer is a nil or invalid reference.	No exception is forwarded or raised if the consumer is a nil or invalid reference.

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Necessary [Ref1]

5.6.10 Interrogable UoF

5.6.10.1 Requirement under Test: SCA426, SCA6

Requirement Text:

SCA426: A BaseComponent shall realize the *ComponentIdentifier* interface.

SCA6: The readonly identifier attribute shall return the instance-unique identifier for a component.

Test Plan Objective/Summary: Ensure a BaseComponent inherits the ComponentIdentifier interface as per realization of the ComponentIdentifier IDL interface. The test procedure obtains an instance of the component and narrows it to that interface to validate the requirement (SCA426). The test procedure also ensure that the identifier is returned by the attribute of the ComponentIdentifier interface (SCA6).

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 45: Steps to execute Test Procedure for SCA426

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ApplicationManager reference is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure “Locate next component or assembly”.	See sub-procedure 6.1.

Step	Action	Expected Result
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the ComponentIdentifier interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the ComponentIdentifier interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Determine that the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contain a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.
9	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::ComponentIdentifier interface. (SCA426)	The componentObject can be narrowed to CF::ComponentIdentifier interface.
11	Retrieve the identifier attribute from the CF::ComponentIdentifier interface and compare it against the value that corresponds to the COMPONENTS_ID id within the specializedInfo field of the application's ComponentType. (SCA6)	The compared values will be equal.
12	Repeat steps 7-11 until no more <componentinstantiation> elements are found within the <componentplacement> element, otherwise go to step 3.	The next componentinstantiation will be evaluated or the verification will go to step 3.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.11 Testable UoF

5.6.11.1 Requirement under Test: SCA428, SCA19, SCA21, SCA23, SCA24, SCA25

Requirement Text:

SCA428: A BaseComponent shall provide a test implementation for all properties whose *kindtype* is "test" as defined in its descriptor files.

SCA19: The *runTest* operation shall use the input testId parameter to determine which of its predefined test implementations should be performed.

SCA21: The *runTest* operation shall return the result(s) of the test in the testValues parameter.

SCA23: The *runTest* operation shall raise the UnknownTest exception when there is no underlying test implementation that is associated with the input testId given.

SCA24: 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.

SCA25: 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.

Test Plan Objective/Summary: Ensure an ApplicationComponent, which is a BaseComponent, provides a test implementation for each test property defined within the component's PRF files. The test procedure extracts the set of test properties defined within the profile and issues the runTest() operation to ensure the component implement those tests. A test is considered implemented (i.e. the runTest uses the testId input parameter to determine the test) if the runTest() operation doesn't raise an UnknownTest exception (SCA428, SCA19, SCA23) . The procedure ensures that resultValues are provided by the test implementation (SCA21). The procedure also checks that a test implementation raises exceptions when invalid properties or values are provided as input values (SCA24, SCA25).

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseFactoryComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.

3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 46: Steps to execute Test Procedure for SCA428, SCA19, SCA21, SCA23, SCA24, SCA25

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure “Locate next component or assembly”.	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure “Obtain supported interface list”	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the TestableInterface interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the TestableInterface interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Perform steps in sub-procedure “Find Component Type of a component instance”.	See sub-procedure 6.2. The ComponentType of the ApplicationComponent is obtained.
9	Obtain the implementation id in the specializedInfo field of the ComponentType of the application component corresponding to the <componentinstantiation> element ID.	The implementation id of the component instance is obtained.
10	Perform steps defined in sub-procedure “Obtain component test properties” with implementation id obtained in step 9 as component implementation id input argument.	See sub-procedure 6.8
11	Obtain the componentObject field of the ComponentType of the application component corresponding to the <componentinstantiation> element ID.	The componentObject is found and the result of the CORBA::is_nil operation on componentObject field is false.

Step	Action	Expected Result
12	Narrow the componentObject to the CF::TestableInterface interface.	The componentObject can be narrowed to CF::TestableInterface interface.
13	Identify the next test from the list of properties obtained in step 10 or go to step 26 when no more test.	The next test to invoke is found or go to step 26.
14	Construct a sequence of properties containing the input values for the testValues inout parameter of the runTest operation using the properties of kind test that are children of the inputvalue specified for the test identified in previous step.	The runTest operation testValues parameter has been constructed.
15	Invoke the runTest() operation on the narrowed componentObject with a testId parameter corresponding to the id attribute of the test property identified in step 13 and the testValues parameter constructed in previous step. (SCA428)(SCA19)(SCA21)(SCA23)	The component recognize the test and the runTest() operation doesn't raise an UnknownTest exception (SCA428)(SCA19) nor an UnknownProperties exception (SCA23).
16	Iterate over the list of properties in the testValues parameter provided by the runTest() operation and compare there ids against the properties of kind test that are children of the resultvalue specified for the test identified in step 13. (SCA21)	The runTest() operation testValues parameter contains all properties specified as child of the resultvalue for the test.
17	Change the value of a property to specify a value outside the range (min – 1 or max + 1) when the sequence of properties constructed in step 14 has a property that support a range, or go to step 20. (SCA24)(SCA25)	A sequence with a valid property but an out of range value is constructed for the runTest() operation testValues parameter or go to step 20.
18	Invoke the runTest() operation on the narrowed componentObject with a testId parameter corresponding to the id attribute of the test property identified in step 13 and the testValues parameter constructed in previous step. (SCA24)(SCA25)	The runtTest() operation raises an UnknownProperties exception. (SCA24)
19	Check the invalidProperties parameter of the UnknownProperties caught in previous step contains the id of the property for which an out of range value has been specified in step 17. (SCA25)	The invalidProperties parameter of the UnknownProperties contains the id of the property for which an out of range value has been specified in step 17.
20	Append the word "Invalid" to a random number, n >= 1, of the properties in the sequence constructed in step 14. (SCA24)(SCA25)	A sequence containing invalid properties is constructed.

Step	Action	Expected Result
21	Invoke the runTest() operation on the narrowed componentObject with a testId parameter corresponding to the id attribute of the test property identified in step 13 and the testValues parameter constructed in previous step. (SCA24)(SCA25)	The runtTest() operation raises an UnknownProperties exception. (SCA24)
22	Check the invalidProperties parameter of the UnknownProperties caught in previous step contains the id of the invalid property(ies) created in step 20. (SCA25)	The invalidProperties parameter of the UnknownProperties contains the id of the property(ies) created in step 20.
23	Repeat steps 13-22 until no more test properties are found within the list.	The next test will be evaluated.
24	Generate a testId that is not present in the list obtained in step 10. (SCA19)(SCA23)	A testId unknown to the component is generated.
25	Invoke the runTest() operation on the narrowed componentObject with the testId generated in previous step. (SCA19)(SCA23)	An UnknownTest exception is raised.
26	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.11.2 Requirement under Test: SCA546

Requirement Text: A BaseComponent shall realize the TestableInterface interface to define and utilize its test properties.

Test Plan Objective/Summary: Ensure an ApplicationComponent, which is a BaseComponent, inherits the TestableInterface interface as per realization of the TestableInterface IDL interface. The test procedure obtains an instance of the component and narrows it to CF::TestableInterface to validate the requirement.

Context:

1. Context Group 3.3: Common Context for requirements associated with BaseComponent (including a sub-application).
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.

2. Precondition Group 4.3: Common Precondition for test procedures requiring an application to be created.
3. This test procedure requires that a BaseComponent implements the Component Registration UoF and that the requirement SCA82 should be verified.

Test Procedure:

Table 47: Steps to execute Test Procedure for SCA546

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps defined in sub-procedure "Obtain supported interface list"	See sub-procedure 6.5.
6	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the TestableInterface interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the TestableInterface interface IDL repository ID or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Determine that the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contain a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.
9	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::TestableInterface interface.	The componentObject can be narrowed to CF::TestableInterface interface.
11	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.6.12 Event Consumer UoF

5.6.12.1 Requirement under Test: SCA444

Requirement Text: A BaseComponent (e.g., ManageableApplicationComponent, DomainManagerComponent, etc.) that consumes events shall implement the CosEventComm::PushConsumer interface.

Test Plan Objective/Summary: Ensure that an application component that consumes events implements the OMG Event Service Specification CosEventComm::PushConsumer interface.

Context:

1. The CosEventComm module is used by consumers for receiving events and by producers for generating events.
2. The CosEventComm interfaces are specified in the OMG Event Service Specification v1.2 (October 2004). https://www.omg.org/spec/EVNT/About-EVNT/
3. It is assumed that the interfaces are implemented via inheritance and it is this inheritance relationship that identifies application components or ports to which this requirement is applicable.
4. An application component that consumes should at least inherit from CosEventComm::PushConsumer defined in the OMG Event Service Specification under the CosEventComm Module.
5. Each source file that implements an application component class should be checked for an inheritance relationship with an interface specified in the OMG Event Service Specification.
6. Each source file that declares or defines an application component which inherits (directly or transitively) from an OMG Event Service Specification interface must be analyzed.
7. The requirement is not applicable to virtual classes. A virtual class is not realizable and therefore cannot produce or generate events.
8. Each source file must be analyzed in the context of full execution of all preprocessor directives.
    - 8.1 All preprocessor-relevant compiler flags are specified
    - 8.2 All included files are available
    - 8.3 All conditional directives are executed and excluded code is eliminated
9. This test does not check for parity between event suppliers and event consumers.

Preconditions:

1. The developer provides the relevant source code files declaring and defining application components.
2. For each source code file, the developer provides all preprocessor-relevant compiler flags and included (including transitively included) files.

Test Procedure:

Table 48: Steps to execute Test Procedure for SCA444

Step	Action	Expected Result
1	Identify source files in scope of SCA444 requirement test.	The source files are identified.
2	Apply preprocessor execution to source files identified in Step 1.	The preprocessor execution directives are applied.
3	For each non-virtual class that is used to implement the BaseComponent, identify those that inherit from CosEventComm::PushConsumer.	The classes that inherit from CosEventComm::PushConsumer are identified.
4	For each class identified in Step 3, verify that the class implements PushConsumer::push() and PushConsumer::disconnect_push_consumer().	The class implements the PushConsumer::push() and PushConsumer::disconnect_push_consumer() methods.
5	Verify that the set of classes identified in Step 3 is non-empty.	The set of classes identified in Step 3 is non-empty.

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Necessary [Ref1]

5.6.13 CORBA Compliant UoF

5.6.13.1 Requirement under Test: SCA506

Requirement Text: Applications shall be limited to using the features designated as mandatory, as specified in E-2.7, for the implemented CORBA profile.

Test Plan Objective/Summary: Ensure an ApplicationComponent does not invoke any of the CORBA functions specified as not required for the SCA CORBA profile the application implements.

Context:

1. The SPD and "source files" referenced in this test case apply to each implemented ApplicationComponent.
2. A "mandatory feature for the SCA CORBA profile the application implements is a CORBA feature or operation listed as MAN under the "Full", "Lightweight" or "Ultra-Lightweight" columns in the tables of document SCA_4.1_App_E-2_Att1_Corba-e.pdf and SCA_4.1_App_E-2_Att2_RT-Corba.pdf.
3. This requirement applies when an ApplicationComponent supports the "sca_compliant" type in its SPD and implements CORBA transport.

4. For an ApplicationComponent to comply with this requirement, its source files may only invoke mandatory features/operations for its declared CORBA profile.
5. A source file which invokes CORBA operations may be conformant with a CORBA profile via conditional compilation or by implementing each CORBA profile in a separate source file without using conditional compilation.
6. If this requirement is verified by source code inspection the following must occur:
   6.1 Each source file which invokes CORBA operations is inspected to ensure only the mandatory CORBA operations defined by the declared CORBA profile are used.
   6.2 When a source file can be used to conform with multiple CORBA profiles via conditional compilation controlled by preprocessor directives, verification is performed in accordance with each profile, only evaluating the code that is available for compilation after preprocessing the directives associated with that profile.
7. If this requirement is verified by compiling application source code and linking with object libraries, the object libraries for the CORBA ORB, SCA Core Framework interfaces and the SCA POSIX AEP functions are provided as described in the "Preconditions" section.
8. The SCA Full CORBA profile does not allow an application to implement the features/operations, designated as not required (NRQ) in SCA_4.1_App_E-2_Att1_Corba-e.pdf and SCA_4.1_App_E-2_Att2_RT-Corba.pdf, listed in the following table.

CORBA Features / Operations Not Allowed in SCA Full CORBA Profile		
Module	**Interface**	**Feature/Operation**
The following are from file SCA_4.1_App_E-2_Att1_Corba-e.pdf.		
CORBA	IDL	Abstract Interfaces
		Value Type
		operation context clauses
		wide character/string
		import
	ORB	id
		get_service_information
		list_initial_services
		register_value_factory
		unregister_value_factory
		lookup_value_factory
		register_initial_reference
	Object	get_interface
		repository_id
		get_component

CORBA Features / Operations Not Allowed in SCA Full CORBA Profile		
Module	**Interface**	**Feature/Operation**
Messaging	Rebind Policy	rebind_mode
	Request End Time Policy	end_time
	Reply End Time Policy	end_time
	Relative Request Timeout Policy	relative_expiry
	Relative Roundtrip Timeout Policy	relative_expiry
Portable Server	Id Uniqueness Policy	value
	POA Manager	get_state
RTCORBA	Mutex	lock
		unlock
		try_lock
	RTORB	create_mutex
		destroy_mutex
CosNaming	NamingContext	[2]
	BindingIterator	[1]
	NamingContextExt	[1]
CosEvent Comm	PullConsumer	[1]
	PullSupplier	[1]
CosEvent Channel Admin	ProxyPushConsumer	[1]
	ProxyPushSupplier	[1]
	ProxyPullConsumer	[1]
	ProxyPullSupplier	[1]
	ConsumerAdmin	[1]
	SupplierAdmin	[1]
[1] No CORBA operations for the interface allowed. [2] No CORBA operations for the interface allowed unless SCA 4.1 Backward Compatibility supported.		

9. The SCA Lightweight CORBA profile does not allow an application to implement CORBA features/operations, designated as not required (NRQ) in SCA_4.1_App_E-2_Att1_Corba-e.pdf and SCA_4.1_App_E-2_Att2_RT-Corba.pdf, listed in the following table.

CORBA Features / Operations Not Allowed in SCA Lightweight CORBA Profile		
Module	**Interface**	**Feature/Operation**
All CORBA features / operations not allowed in the SCA Full CORBA profile.		
The following are from file SCA_4.1_App_E-2_Att1_Corba-e.pdf.		
CORBA	IDL	any
	ORB	resolve_initial_references
		create_policy
	Object	is_nil
		duplicate
		release
		non_existent
		is_equivalent
		get_policy
		set_policy_overrides
		get_client_policy
		get_policy_overrides
		get_orb
	Policy	policy_type
		copy
		destroy
	PolicyManager	get_policy_overrides
		set_policy_overrides
	PolicyManager	[1]
	Type Code	[1]
	Policy Current	[1]
	Sync Scope Policy	[1]
The following are from file SCA_4.1_App_E-2_Att2_RT-Corba.pdf.		
RTCORBA	Priority Mode Policy	SERVER_DECLARED
	Thread Pools	create_threadpool
		create_threadpool_with_lanes

CORBA Features / Operations Not Allowed in SCA Lightweight CORBA Profile		
Module	**Interface**	**Feature/Operation**
	POA	activate_object_with_priority
[1] No CORBA operations for the interface allowed.		

10. The SCA Ultra-Lightweight CORBA profile does not allow an application to implement CORBA features/operations, designated as not required (NRQ) in SCA_4.1_App_E-2_Att1_Corba-e.pdf, listed in the following table.

CORBA Features / Operations Not Allowed in SCA Ultra-Lightweight CORBA Profile		
Module	**Interface**	**Feature/Operation**
All CORBA features / operations not allowed in the SCA Full and Lightweight CORBA profiles.		
The following are from file SCA_4.1_App_E-2_Att1_Corba-e.pdf.		
CORBA	IDL	float, double, long double, long long, unsigned long long, char, string
		unions
		arrays
		IDL basic data types other than boolean, octet, short, unsigned short, long, unsigned long and enum are not allowed
		IDL keywords other than module, interface, in, out, inout, void, typedef, oneway are not allowed
	ORB	orb_init
		object_to_string
		string_to_object
		work_pending
		perform_work
		run
		shutdown
		destroy
	Object	is_a
		validate_connection
	POA	the_POAManager
		activate_object
		activate_object_with_id
		deactivate_object
CosNaming	NamingContext	[1]

[1] No CORBA operations for the interface allowed. SCA 4.1 Backwards Compatibility not supported.

Preconditions:

1. Precondition Group 4.2: Common Precondition for test procedures involving parsing of XML files.
2. The developer specifies the source file names used to build each ApplicationComponent executable.
3. A CORBA ORB function and symbol resolution object file exists on the test platform for each CORBA profile. This may be a fully functional implementation of the mandatory CORBA functions and symbols for a specific CORBA profile or may only be the CORBA constants and function signatures, and if applicable, a default return value in the function body, for each of the mandatory CORBA functions and symbols.
4. An SCA CF function and symbol resolution object file may exist on the test platform. This may be a fully functional SCA CF implementation generated by a CORBA IDL to CPP compiler or only the SCA constants and function signatures, and if applicable, a default return value in the function body, for each of the CF interfaces.
5. An RTOS symbol resolution object file exists on the test platform for each SCA POSIX AEP profile. This may be a fully functional implementation of the mandatory OS services for a specific AEP profile or may only be the AEP constants and function signatures, and if applicable, a default return value in the function body, for each of the mandatory OS services.

Test Procedure:

Table 49: Steps to execute Test Procedure for SCA506

Step	Action	Expected Result
1	Locate all the ApplicationComponents to which the test procedure applies.	A list of ApplicationComponents to which the test procedure applies is identified.
2	Compile the source files of the next ApplicationComponent in the list.	The source files are successfully compiled and the resulting object files are saved on the test platform.
3	Link the ApplicationComponent's object files with the CORBA symbol resolution object file for the target CORBA profile, and if present, the CF and the OS services symbol resolution object files.	The linker reports all external function references that it cannot resolve, which may be non-compliant CORBA symbols (functions, exceptions, constants). If the CF and OS services symbol resolution object files are not present, the linker will report external CF and OS service function and constant references it cannot resolve.
4	Evaluate the linker output error messages to determine if they reference non-required CORBA symbols (functions, exceptions, constants) for the CORBA profile being evaluated for compliance. Disregard linker error messages that do not reference CORBA symbols, such as SCA CF interfaces and constants, OS service functions and constants.	If there are no linker error messages referencing non-required CORBA symbols for the CORBA profile being evaluated for compliance, the ApplicationComponent conforms to the CORBA profile.
5	Repeat steps 2-4 until all CORBA profiles for the ApplicationComponent to be inspected for CORBA compliance have been evaluated.	The next CORBA profile for the ApplicationComponent is evaluated or continue with step 6.
6	Continue with step 2 for the next ApplicationComponent to be evaluated for CORBA compliance.	The next ApplicationComponent is evaluated for CORBA compliance or the test terminates.

Postconditions: N/A

Test Plan Verification Method: Analysis [Ref1]

Test Plan Result Category: Necessary [Ref1]

If the test passes, the ApplicationComponent does not invoke any of the non-required CORBA functions stated in the test procedure. The test does not confirm the ApplicationComponent invokes only mandatory CORBA functions.

5.7 BaseFactoryComponent Test Procedures

5.7.1 Mandatory

5.7.1.1 Requirement under Test: SCA386, SCA387, SCA388, SCA415

Requirement Text:

SCA386: The *createComponent* operation shall create a component if no component exists for the given componentId.

SCA387: The *createComponent* operation shall assign the given componentId to a new component.

SCA388: The *createComponent* operation shall return a CF::ComponentType structure.

SCA415: The ApplicationComponentFactoryComponent shall only deploy ApplicationComponents.

Test Plan Objective/Summary: Ensure an ApplicationComponentFactoryComponent, which is a BaseFactoryComponent, can create a component for which a given componentId has not already been used to create a component. The returned ComponentType structure is inspected to verify it contains the appropriate information to satisfy each requirement. The test procedure attempts to create a component by using a componentId that does not exist, in conjunction with valid factoryparam property IDs and values.

Context:

1. Context Group 3.9: Common Context for requirements associated with a component which implements the ComponentFactory interface.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
4. The SCD of a component created by a ComponentFactory can be found by parsing the SPD file associated with the component that referenced the factory in its <componentfactoryref> child element (SCA415).

Preconditions:

1. Precondition Group 4.5: Common Precondition for test procedures involving execution of a BaseFactoryComponent (an ApplicationComponentFactoryComponent is a BaseFactoryComponent).

Test Procedure:

Table 50: Steps to execute Test Procedure for SCA386, SCA387, SCA388, SCA415

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Identify the next <componentplacement> element within the <partitioning> element.	The location of the <componentplacement> element is identified or the verification will terminate.
4	Perform steps in sub-procedure "Obtain type of a component placement".	See sub-procedure 6.3.
5	Verify that the value of the <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT.	The value of <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT or go back to step 3.
6	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
7	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2.
8	Perform steps in sub-procedure "Obtain factoryparam properties for a component created by a component factory".	See sub-procedure 6.4
9	Generate a random string value for the componentId and ensure that it is not referenced in the ApplicationManager's ComponentType.	The identifier is not defined within the application.
10	Invoke the createComponent operation on the componentObject (narrowed into CF::ComponentFactory) obtained in step 7 with the componentId parameter and the Properties sequence containing the factoryparam properties obtained in step 8 for the qualifiers parameter.(SCA386) (SCA388)	The createComponent operation returns a ComponentType structure for which the result of the CORBA::is_nil operation returns false.
11	Verify that the identifier field of the ComponentType structure returned by the createComponent operation is equal to the componentId value provided in step 9. (SCA387) (SCA388)	The identifier field of the Component structure is equal to the componentId provided in step 9.

Step	Action	Expected Result
12	Verify that the profile field of the ComponentType structure returned by the createComponent operation indicates the component's profile filename or the profile itself (can be verified checking against the value of the <componentfile> associated with the <componentplacement> element that is the parent of the <componentinstantiation> element found in step 6). (SCA388)	The profile field of the Component structure indicates the component's profile filename or the profile itself.
13	Verify that the type field of the ComponentType structure returned by the createComponent operation is a value mapping to the of the component's SCD <componenttype> element as described in Appendix D-1. (SCA388)	The type field of the Component structure maps to the value of the <componenttype> element of the component's SCD.
14	Verify that the type field of the ComponentType structure returned by the createComponent operation is either an APPLICATION_COMPONENT, MANAGEABLE_APPLICATION_COMPONENT or APPLICATION_COMPONENT_FACTORY_COMPONENT (SCA415)	The type field of the Component structure maps to a component type that is APPLICATION_COMPONENT, MANAGEABLE_APPLICATION_COMPONENT or APPLICATION_COMPONENT_FACTORY_COMPONENT.
15	Verify that the result of the CORBA::is_nil operation returns false for the componentObject field of the ComponentType structure returned by the createComponent. (SCA388)	The result of the CORBA::is_nil operation returns false for the componentObject field Component structure.
16	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.7.1.2 Requirement under Test: SCA389

Requirement Text: The *createComponent* operation shall raise the CreateComponentFailure exception when it cannot create the component or the component already exists.

Test Plan Objective/Summary: Ensure an ApplicationComponentFactoryComponent, which is a BaseFactoryComponent, raises the CreateComponentFailure exception when it cannot create a component or when the component to create already exists. For the first part of the requirement, the test procedure attempts to create component by using a componentId that does not exist, in conjunction with invalid factoryparam property IDs and values. For the second part of the requirement, the test procedure attempts to create a component using an existing componentId.

Context:

1. Context Group 3.9: Common Context for requirements associated with a component which implements the ComponentFactory interface.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.
4. It is assumed that a ComponentFactory doesn't ignore a factoryparam property that it doesn't know and raises an exception when one is provided when it is asked to create a component.

Preconditions:

1. Precondition Group 4.5: Common Precondition for test procedures involving execution of a BaseFactoryComponent (an ApplicationComponentFactoryComponent is a BaseFactoryComponent).

Test Procedure:

Table 51: Steps to execute Test Procedure for SCA389

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Identify the next <componentplacement> element within the <partitioning> element.	The location of the <componentplacement> element is identified or the verification will terminate.
4	Perform steps in sub-procedure "Obtain type of a component placement".	See sub-procedure 6.3.
5	Verify that the value of the <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT.	The value of <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT or go back to step 3.
6	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.

Step	Action	Expected Result
7	Perform steps in sub-procedure “Find Component Type of a component instance”.	See sub-procedure 6.2.
8	Generate a random string value for the componentId and ensure that it is not referenced in the ApplicationManager’s ComponentType.	The identifier is not defined within the application.
9	Create a Properties sequence containing factoryparam properties that have arbitrary id and value such that the id is unknown to the ApplicationFactoryComponentFactory.	A sequence of factoryparam parameters unknown to the ApplicationFactoryComponentFactory is created.
10	Invoke the createComponent operation on the componentObject (narrowed into CF::ComponentFactory) obtained in step 7 with the componentId parameter created in step 8 and the Properties sequence containing the factoryparam properties created in step 9.	The createComponent operation raises a CreateComponentFailure exception.
11	Perform steps in sub-procedure “Obtain factoryparam properties for a component created by a component factory”.	See sub-procedure 6.4
12	Invoke the createComponent operation on the componentObject (narrowed into CF::ComponentFactory) obtained in step 7 with the id of the <componentinstantiation> element found in step 11 for the componentId parameter and the Properties sequence containing the factoryparam properties obtained in step 11 for the qualifiers parameter.	The createComponent operation raises a CreateComponentFailure exception.
13	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Necessary [Ref1]

The test may return a false positive for an undefined factoryparam property. In that case, source code inspection should be done.

5.7.1.3 Requirement under Test: SCA413, SCA549

Requirement Text:

SCA413: A BaseFactoryComponent shall realize the ComponentFactory interface.

SCA549: A BaseFactoryComponent shall realize the *LifeCycle* interface.

Test Plan Objective/Summary: Ensure an ApplicationComponentFactoryComponent, which is a BaseFactoryComponent, inherits the ComponentFactory interface and the LifeCycle interface as per realization of the ComponentFactory and LifeCycle IDL interfaces. The test procedures obtain an instance of the component and narrow it to the appropriate interface depending on the requirement (SCA413 or SCA549).

Context:

1. Context Group 3.7: Common Context for requirements associated with BaseFactoryComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.5: Common Precondition for test procedures involving execution of a BaseFactoryComponent.

Test Procedure:

Table 52: Steps to execute Test Procedure for SCA413 and SCA549

Step	Action	Expected Result
1	Obtain the ComponentType struct of the ApplicationManager instance of the application under test.	The ComponentType struct is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Identify the next <componentplacement> element within the <partitioning> element.	The location of the <componentplacement> element is identified or the verification will terminate.
4	Perform steps defined in sub-procedure "Obtain type of a component placement"	See sub-procedure 6.3.
5	Verify that the value of the <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT.	The value of <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT or go back to step 3.
6	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
7	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2.
8	Obtain the componentObject field of the ComponentType.	The result of the CORBA::is_nil operation on componentObject field is false.
9	Narrow the componentObject field of the ComponentType reference to the CF:ComponentFactory interface. (SCA413)	The result of the CORBA::is_nil operation on the narrowed componentObject field is false.
10	Narrow the componentObject field of the ComponentType reference to the CF:LifeCycle interface. (SCA549)	The result of the CORBA::is_nil operation on the narrowed componentObject field is false.
11	Return to step 3.	The next component will be evaluated.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.7.1.4 Requirement under Test: SCA414

Requirement Text: A BaseFactoryComponent shall fulfill the BaseComponent requirements.

Test Plan Objective/Summary: Ensure that a BaseFactoryComponent fulfills all requirements of a BaseComponent by invoking all test procedures associated with BaseComponent for that BaseFactoryComponent.

Context:

1. Context Group 3.7: Common Context for requirements associated with BaseFactoryComponent.

Preconditions: N/A

Test Procedure:

Table 53: Steps to execute Test Procedure for SCA414

Step	Action	Expected Result
1	Execute the test procedures of the following requirements: SCA427, SCA430, SCA548, SCA463, SCA501, SCA502, SCA503, SCA494, SCA495, SCA420, SCA421, SCA423, SCA429, SCA545, SCA26, SCA28, SCA29, SCA30, SCA31, SCA432, SCA15, SCA518, SCA16, SCA17, SCA18, SCA433, SCA32, SCA33, SCA34, SCA36, SCA37, SCA547, SCA7, SCA519, SCA8, SCA10, SCA11, SCA12, SCA13, SCA14, SCA424, SCA425, SCA444, SCA426, SCA6, SCA428, SCA546, SCA19, SCA21, SCA23, SCA24, SCA25 of the application under test relative to its incorporated Units of Functionality.	The application is validated to have an implementation that complies with the set of BaseComponent requirements identified by its Units of Functionality.

Postconditions: N/A

Test Plan Verification Method: N/A (The verification method of individual test procedures will apply)
Test Plan Result Category: N/A (The result category of individual test procedures will apply)

5.7.1.5 Requirement under Test: SCA540

Requirement Text: Each BaseFactoryComponent shall support the mandatory Component Identifier execute parameter as described in section 3.1.3.3.1.3.5.1, in addition to their user-defined execute properties in the component's SPD.

Test Plan Objective/Summary: Ensure an ApplicationComponentFactoryComponent, which is a BaseFactoryComponent, supplies the component identifier provided by a mandatory executable parameter into its ComponentType structure. The ComponentType structure returned by an application contains a specializedInfo field identified by an id of COMPONENTS_ID and a type CF::Components. One element of the CF::Components must have an identifier equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.

Context:

1. Context Group 3.7: Common Context for requirements associated with BaseFactoryComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.5: Common Precondition for test procedures involving execution of a BaseFactoryComponent.

Test Procedure:

Table 54: Steps to execute Test Procedure for SCA540

Step	Action	Expected Result
1	Obtain the ComponentType struct of the ApplicationManager instance of the application under test.	The ComponentType struct is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps in sub-procedure "Obtain type of a component placement".	See sub-procedure 6.3.
6	Verify that the value of the <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT.	The value of <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT or go back to step 3.
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2.
9	Repeat steps 7-8 until no more <componentinstantiation> elements are found within the <componentplacement> element, otherwise go to step 3.	The next componentinstantiation will be evaluated or the verification will go to step 3.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.7.2 Interrogable UoF

5.7.2.1 Requirement under Test: SCA541

Requirement Text: Each executable BaseFactoryComponent shall set its identifier attribute using the Component Identifier execute parameter.

Test Plan Objective/Summary: Ensure an ApplicationComponentFactoryComponent, which is a BaseFactoryComponent, has a component identifier attribute value that is equivalent to that stored within the component's ComponentType structure. The ComponentType structure returned by an application contains a specializedInfo field identified by an id of COMPONENTS_ID and a type CF::Components. One element of the CF::Components must have an identifier equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component. The test procedure obtains an instance of the component and narrows it to CF::ComponentIdentifier to validate the requirement.

Context:

1. Context Group 3.7: Common Context for requirements associated with BaseFactoryComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.5: Common Precondition for test procedures involving execution of a BaseFactoryComponent.
2. This test procedure requires that an ApplicationComponentFactoryComponent implements the Component Registration UoF and that the requirement SCA540 has been verified.
3. This test procedure requires that the requirement SCA540 has been verified prior.

Test Procedure:

Table 55: Steps to execute Test Procedure for SCA541

Step	Action	Expected Result
1	Obtain the ComponentType struct of the ApplicationManager instance of the application under test.	The ComponentType struct is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure "Locate next component or assembly".	See sub-procedure 6.1.
4	Perform steps defined in sub-procedure "Obtain type of a component placement".	See sub-procedure 6.3.
5	Verify that the value of the <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT.	The value of <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT or go back to step 3.
6	Perform steps defined in sub-procedure "Obtain supported interface list".	See sub-procedure 6.5.
7	Verify that the <componentfeatures> element contains a <supportsinterface> element with a repid attribute equals to the ComponentIdentifier interface IDL repository ID.	The <componentfeatures> contains a <supportsinterface> element with a repid attribute equals to the ComponentIdentifier interface IDL repository ID or go back to step 3.
8	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
9	Perform steps in sub-procedure "Find Component Type of a component instance".	See sub-procedure 6.2.
10	Retrieve the identifier attribute from the factory component's ComponentIdentifier interface and compare it against the value that corresponds to the COMPONENTS_ID id within the specializedInfo field of the application's ComponentType.	The compared values will be equal or the verification procedure will fail.
13	Repeat steps 8-10 until no more <componentinstantiation> elements are found within the <componentplacement> element, otherwise go to step 3.	The next componentinstantiation will be evaluated or the verification will go to step 3.

Postconditions: N/A

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

5.7.3 Releaseable UoF

5.7.3.1 Requirement under Test: SCA574

Requirement Text: The *releaseObject* operation shall release all component instances created by the BaseFactoryComponent.

Test Plan Objective/Summary: Ensure all components created by a BaseFactoryComponent are no longer usable after the BaseFactoryComponent has been released. The test procedure obtains the instances of the components before the BaseFactoryComponent is released and try to invoke an operation after the BaseFactoryComponent is released.

Context:

1. Context Group 3.7: Common Context for requirements associated with BaseFactoryComponent.
2. Context Group 3.2: Common Context related to the validation of an application component.
3. Context Group 3.1: Common Context for SCA 4.1 Test Platform.

Preconditions:

1. Precondition Group 4.5: Common Precondition for test procedures involving execution of a BaseFactoryComponent.

Test Procedure:

Table 56: Steps to execute Test Procedure for SCA574

Step	Action	Expected Result
1	Obtain the ComponentType reference of the ApplicationManager instance of the application under test.	The ComponentType reference of the ApplicationManager is obtained.
2	Locate the <partitioning> element within the SAD.	The <partitioning> element is found within the SAD.
3	Perform steps defined in sub-procedure “Locate next component or assembly”.	See sub-procedure 6.1.
4	Go to step 5 if the element being evaluated is a <componentplacement> or return to step 2 and proceed with the file identified in the <assemblyplacement>.	The next child element of the current element will be evaluated or the verification terminate.
5	Perform steps in sub-procedure “Obtain type of a component placement”.	See sub-procedure 6.3.
6	Verify that the value of the <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT.	The value of <componenttype> element is APPLICATION_COMPONENT_FACTORY_COMPONENT or go back to step 3.

Step	Action	Expected Result
7	Identify in the SAD the next <componentinstantiation> element within the current <componentplacement> element.	The location of the <componentinstantiation> element is identified.
8	Perform steps in sub-procedure “Find Component Type of a component instance”.	See sub-procedure 6.2. The ComponentType of the BaseFactoryComponent is obtained.
9	Obtain the componentObject field of the ComponentType of the application component factory component corresponding to the <componentinstantiation> element ID.	The componentObject is found and the result of the CORBA::is_nil operation on componentObject field is false.
10	Narrow the componentObject to the CF::ComponentFactory interface.	The componentObject can be narrowed to CF::ComponentFactory interface.
11	Invoke the releaseObject() operation on the narrowed componentObject.	The BaseFactoryComponent is released.
12	Parse the SAD to find the next <componentinstantiation> element that has a <componentfactoryref> child element with the value of its refid attribute equal to the id of <componentinstantiation> element being processed.	A <componentinstantiation> is found within the assembly that corresponds to the factory reference being processed.
13	Perform steps in sub-procedure “Find Component Type of a component instance”.	See sub-procedure 6.2. The ComponentType of an application component created by the BaseFactoryComponent is obtained.
14	Obtain the componentObject field of the ComponentType of the application component corresponding to the <componentinstantiation> element ID.	The componentObject is found and the result of the CORBA::is_nil operation on componentObject field is false.
15	Using the componentObject for the component, invoke CORBA::is_a("IDL:CF/LifeCycle:1.0").	CORBA::is_a() throws a OBJECT_NOT_EXIST CORBA system exception.
16	Repeat steps 12-15 until no more <componentinstantiation> elements that has a <componentfactoryref> child element with the value of its refid attribute equal to the id of <componentinstantiation> element of the BaseFactoryComponent being processed are found.	The next componentinstantiation of an application component created by the BaseFactoryComponent will be evaluated or the verification will go to step 17.
17	Return to step 3.	The next component will be evaluated.

Postconditions: The test environment needs to be reset because the ApplicationManagerComponent is still executing.

Test Plan Verification Method: Test [Ref1]

Test Plan Result Category: Sufficient and Necessary [Ref1]

6 Reusable Verification Sub-procedures

6.1 Locate next component or assembly

Sub-procedure Objective/Summary: Traverse the SAD to find the next componentplacement or assemblyplacement sub-element in the partitioning element that describes a component or nested application. In the sub-element found, find the <componentfile> element and verify that the file it references exists on the test platform.

Table 57: Steps to execute Sub-procedure 6.1 Locate next component or assembly

Step	Action	Expected Result
1	Identify the next <componentplacement> or <assemblyplacement> element within the <partitioning> element.	The location of the <componentplacement> or <assemblyplacement> element is identified.
2	Determine if the file identified by <componentfile> element referenced by the <componentplacement> or <assemblyplacement> element exists.	The file identified by the <componentfile> reference within the <partitioning> subelement exists.

6.2 Find Component Type of a component instance

Sub-procedure Objective/Summary: Validate that the deployed application that is processed and evaluated by the test procedure contains a component with an identifier equal to the identifier passed to the application component as an executable parameter.

Table 58: Steps to execute Sub-procedure 6.2 Find Component Type of a component instance

Step	Action	Expected Result
1	Determine if the ComponentType returned by the application contains a specializedInfo field identified by an id of COMPONENTS_ID, which in turn contains a ComponentType for which the identifier field is equal to the value of the COMPONENT_IDENTIFIER execute parameter received by the component.	The ApplicationManager's ComponentType has within the ComponentType sequence contained in the value of its specializedInfo field identified with an ID COMPONENTS_ID a ComponentType for which the identifier field is equal to the value of the <componentinstantiation> element's id attribute followed by ":" and the ApplicationManager name.

6.3 Obtain type of a component placement

Sub-procedure Objective/Summary: Identify the type of component referenced by a component (identified by a componentplacement element) being evaluated. The type is obtained by extracting the componenttype element value from the SCD file associated with the referenced SPD.

Table 59: Steps to execute Sub-procedure 6.3 Obtain type of a component placement

Step	Action	Expected Result
1	Determine if the file identified by <componentfile> element referenced by the <componentplacement> element exists.	The file identified by the <componentfile> reference within the <componentplacement> element exists.
2	Open the SPD file indicated by the <componentfile> referenced by the <componentplacement> and locate the <descriptor> element.	The <descriptor> element is located or verification will go back to the step prior to the invocation of this subprocedure.
3	Determine if the file identified by <localfile> element referenced by the <descriptor> element exists.	The file identified by the <localfile> reference within the <descriptor> subelement exists.
4	Open the SCD file indicated by the <descriptor> element and locate the <componenttype> element.	The SCD file can be opened and <componenttype> element is found.
5	Obtain the value of the <componenttype> element.	The value of the <componenttype> element is obtained.

6.4 Obtain factoryparam properties for a component created by a component factory

Sub-procedure Objective/Summary: Locate an application component within the SAD which is instantiated by the application component factory component which is being evaluated. Retrieve the factory properties for that component instantiation and their values in accordance with the property location precedence strategy defined within SCA appendix D.

Table 60: Steps to execute Sub-procedure 6.4 Obtain factoryparam properties for a component created by a component factory

Step	Action	Expected Result
1	Parse the SAD to find a <componentinstantiation> element that has a <componentfactoryref> child element with the value of its refid attribute equal to the id of <componentinstantiation> element being processed.	A <componentinstantiation> is found within the assembly that corresponds to the factory reference being processed.
2	Obtain the <factoryparam> property IDs and their final values for the component associated with the <componentinstantiation> element found in step 1 by parsing the Domain Profile and applying the precedence order defined in section D-1.10.1.3.1.2.1 of Appendix D-1.	The factoryparam property IDs and their values are obtained.

6.5 Obtain supported interface list

Sub-procedure Objective/Summary: Identify the supported interfaces associated with the component (identified by a componentplacement element) being evaluated. The supported interfaces are obtained by extracting the values contained within the supportsinterface element(s) from the SCD file associated with the referenced SPD.

Table 61: Steps to execute Sub-procedure 6.5 Obtain supported interface list

Step	Action	Expected Result
1	Determine if the file identified by the <componentfile> element referenced by the <componentplacement> element exists.	The file identified by the <componentfile> reference within the <componentplacement> element exists.
2	Open the SPD file indicated by the <componentfile> referenced by the <componentplacement> and locate the <descriptor> element.	The <descriptor> element is located or verification will go back to the step prior to the invocation of this subprocedure.
3	Determine if the file identified by the <localfile> element referenced by the <descriptor> element exists.	The file identified by the <localfile> reference within the <descriptor> subelement exists.
4	Open the SCD file indicated by the <descriptor> element and locate the <componentfeatures> element.	The SCD file can be opened and the <componentfeatures> element is found.
5	Obtain the value of the list of <supportsinterface> elements.	The list of <supportsinterface> element is obtained.

6.6 Ensure an operation raises a specific exception

Sub-procedure Objective/Summary: Locate and open the source code file associated with the component under evaluation. Search the code to confirm that the code for the operation identified by the operation name input parameter raises the exception identified by the exception name input parameter for the set of error conditions associated with the operation name and exception name input parameters.

Sub-procedure input parameters: SCA operation name and exception name.

Table 62: Steps to execute Sub-procedure 6.6 Ensure an operation raises a specific exception

Step	Action	Expected Result
1	Determine the source file names associated with the SPD for the BaseComponent (see Preconditions for this information).	The source file names associated with the SPD for the BaseComponent are determined.
2	Search the source files of the BaseComponent for the specified input SCA operation name and verify the operation raises the specified input exception name if an error occurs.	The input SCA operation is found and the fact that the operation raises the specified input exception if an error occurs is verified

6.7 Obtain component configure properties

Sub-procedure Objective/Summary: Locate and open all of the property files referenced within the application's domain profile for the implementation specified by the input implementation id parameter. Construct a union of the application component's configure properties which match the property modes identified by the input property mode parameter.

Sub-procedure input parameters: Component implementation id and valid property modes (readonly and readwrite, writeonly and readwrite).

Table 63: Steps to execute Sub-procedure 6.7 Obtain component configure properties

Step	Action	Expected Result
1	Locate the SPD file for the ApplicationComponent.	The SPD file for the ApplicationComponent is found.
2	Locate the SCD file for the ApplicationComponent, which is defined in the <descriptor> element in the SPD.	The SCD file for the ApplicationComponent is found.
3	Locate all the PRF files for the ApplicationComponent which may be referenced by one or more of the following: 1. In the SPD, in the <propertyfile> in the <softpkg>. 2. In the SPD, in the <propertyfile> in the <implementation> deployed on the verification platform (the one that matches the component implementation id input parameter). 3. In the SCD, in the <propertyfile> in the <softwarecomponent>.	All the PRF files for the ApplicationComponent are found.
4	Compile a list of all unique properties with kind configure and mode constrained by the valid property modes input parameter for the implementation specified by the component implementation id input parameter.	A list of all property information for the ApplicationComponent is constructed.

6.8 Obtain component test properties

Sub-procedure Objective/Summary: Locate and open all of the property files referenced within the application's domain profile for the implementation specified by the input implementation id parameter. Construct a union of the application component's test properties.

Sub-procedure input parameters: Component implementation id.

Table 64: Steps to execute Sub-procedure 6.8 Obtain component test properties

Step	Action	Expected Result
1	Locate the SPD file for the ApplicationComponent.	The SPD file for the ApplicationComponent is found.
2	Locate the SCD file for the ApplicationComponent, which is defined in the <descriptor> element in the SPD.	The SCD file for the ApplicationComponent is found.
3	Locate all the PRF files for the ApplicationComponent which may be referenced by one or more of the following: 1. In the SPD, in the <propertyfile> in the <softpkg>. 2. In the SPD, in the <propertyfile> in the <implementation> deployed on the verification platform (the one that matches the component implementation id input parameter). 3. In the SCD, in the <propertyfile> in the <softwarecomponent>.	All PRF files of the ApplicationComponent are found.
4	Compile a list of all unique test properties (test element) for the implementation specified by the component implementation id input parameter.	A list of all property information for the ApplicationComponent, including the input and result values, is constructed.

7 References

[Ref0] : Software Communications Architecture Specification, Version 4.1, 20 August 2015
[Ref1] : SCA4.1 Applications Verification Plan, WINNF-TR-4001, 21 February 2018

Appendix A

Table 65 provides an overview of when requirements are applicable to a component under test. The table is separated by a thick black vertical line. The left part indicates the application component type(s) for which a requirement applies. The requirement is applicable to any type of component marked with an X. The right part indicates the unit of functionality (UoF) implementations that are necessary for a requirement to be applicable. When no UoF is indicated, it means the requirement is mandatory for the type of component. When requirement applicability depends on component implementation of a UoF, the UoF(s) are marked with an X.

Table 65: Overview of when a requirement is applicable.

Requirement	Application Component	Manageable Application Component	Application Controller Component	Application Component Factory Component	Assembly Component	Interrogable	LifeCycle	Releaseable	Configurable	Controllable	Component Registration	Connectable	Log Producer	Event Producer	Event Consumer	Testable	Channel Extension	CORBA Compliant
SCA386				X														
SCA387				X														
SCA388				X														
SCA389				X														
SCA427	X	X	X	X	X													
SCA430	X	X	X	X														
SCA548	X	X	X	X														
SCA540				X														
SCA413				X														
SCA414				X														
SCA549				X														
SCA169		X																
SCA173	X	X	X	X														
SCA457	X	X	X	X														
SCA551	X	X	X	X														
SCA455		X																

Requirement	Application Component	Manageable Application Component	Application Controller Component	Application Component Factory Component	Assembly Component	Interrogable	LifeCycle	Releaseable	Configurable	Controllable	Component Registration	Connectable	Log Producer	Event Producer	Event Consumer	Testable	Channel Extension	CORBA Compliant
SCA456		X																
SCA520		X																
SCA166		X																
SCA167		X																
SCA550		X																
SCA175			X															
SCA176			X															
SCA415				X														
SCA521				X														
SCA522				X														
SCA155					X													
SCA156					X													
SCA463	X	X	X	X	X													
SCA471	-	-	-	-	-													
SCA501	X	X	X	X	X													
SCA502	X	X	X	X	X													
SCA496			X															
SCA503	X	X	X	X														
SCA494	X	X	X	X														
SCA495	X	X	X	X														
SCA420	X	X	X	X					X				X					
SCA421	X	X	X	X									X					
SCA423	X	X	X	X									X					
SCA429	X	X	X	X					X									
SCA545	X	X	X	X					X									

Requirement	Application Component	Manageable Application Component	Application Controller Component	Application Component Factory Component	Assembly Component	Interrogable	LifeCycle	Releaseable	Configurable	Controllable	Component Registration	Connectable	Log Producer	Event Producer	Event Consumer	Testable	Channel Extension	CORBA Compliant
SCA26	X	X	X	X					X									
SCA27	X	X	X	X					X									
SCA28	X	X	X	X					X									
SCA29	X	X	X	X					X									
SCA30	X	X	X	X					X									
SCA31	X	X	X	X					X									
SCA432	X	X	X	X			X											
SCA15	X	X	X	X			X											
SCA518	X	X	X	X				X										
SCA574				X				X										
SCA16	X	X	X	X				X										
SCA17	X	X	X	X				X										
SCA18	X	X	X	X				X										
SCA433	X	X	X	X						X								
SCA32	X	X	X	X						X								
SCA33	X	X	X	X						X								
SCA34	X	X	X	X						X								
SCA36	X	X	X	X						X								
SCA37	X	X	X	X						X								
SCA547	X	X	X	X								X						
SCA7	X	X	X	X								X						
SCA519	X	X	X	X								X						
SCA8	X	X	X	X								X						
SCA10	X	X	X	X								X						
SCA11	X	X	X	X								X						

Requirement	Application Component	Manageable Application Component	Application Controller Component	Application Component Factory Component	Assembly Component	Interrogable	LifeCycle	Releaseable	Configurable	Controllable	Component Registration	Connectable	Log Producer	Event Producer	Event Consumer	Testable	Channel Extension	CORBA Compliant
SCA12	X	X	X	X								X						
SCA13	X	X	X	X								X						
SCA14	X	X	X	X								X						
SCA82	X	X	X	X							X							
SCA424	X	X	X	X										X				
SCA425	X	X	X	X										X				
SCA444	X	X	X	X											X			
SCA426	X	X	X	X		X												
SCA541				X		X												
SCA6	X	X	X	X		X												
SCA168		X	X			X												
SCA428	X	X	X	X												X		
SCA546	X	X	X	X												X		
SCA19	X	X	X	X												X		
SCA21	X	X	X	X												X		
SCA23	X	X	X	X												X		
SCA24	X	X	X	X												X		
SCA25	X	X	X	X												X		
SCA500			X														X	
SCA506	X	X	X	X														X