Joint Tactical Networking Center Test and Evaluation Laboratory

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

Appendix B Volume 4
Test Procedures for Framework Services Interfaces

13 April 2022

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

Prepared by:
Joint Tactical Networking Center Test and Evaluation Laboratory

TABLE OF CONTENTS

APPENDIX B VOLUME 4 TEST PROCEDURES

B.4. Volume 4 Framework Services Interfaces

This volume consists of manual test cases, which verify requirements related to Software Communication Architecture (SCA) Framework Services Interfaces.

B.4.1. OE_TC_031 - FileSystem :: list

Test Case Number: OE_TC_031

FileSystem::list

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0528	At a minimum, the file system shall support name, kind, and size information for a file.
OE0542	The *list* operation shall support the "*" and "?" wildcard characters (used to match any sequence of characters (including null) and any single character, respectively.
OE0543	The *list* operation shall return a FileInformationSequence for files that match the search pattern specified in the input pattern parameter.
OE0582	A file manager shall implement the inherited *FileSystem* operations as required under section 3.1.3.4.2 for each mounted file system.
OE0583	The *FileSystem* operations inherited by a file manager shall remove the name of the mounted file system from input pathnames before passing the pathnames to any operation on a mounted file system.
OE0736	These wildcards shall only be applied following the right-most forward-slash character ("/") in the pathname contained in the input pattern parameter.
OE0755	The *list* operation shall return a zero length sequence when no file is found which matches the search pattern.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-80, Section 3.1.3.4.2.3.3; Page 3-83, Section 3.1.3.4.2.5.4.3; Page 3-83, Section 3.1.3.4.2.5.4.4; Page 3-89, Section 3.1.3.4.3.5.4; Page 3-89, Section 3.1.3.4.3.5.4; Page 3-83, Section 3.1.3.4.2.5.4.3; Page 3-83, Section 3.1.3.4.2.5.4.4

Test Objective

This test case verifies OE0528, OE0542, OE0543, OE0582, OE0583, OE0736, and OE0755 for a *FileManager*. The objective of this test is to verify that the *FileManager* correctly implements or delegates the *list* operation to the mounted *FileSystems* (OE0582) when supporting the "*" and "?" wildcard characters (OE0542) in the search parameter, checking that certain other requirements are also fulfilled (OE0528, OE0543, OE0583, OE0736, OE0755). The *FileManager*'s *list* operation must return a FileInformationSequence matching the search pattern specified in the input parameter (OE0543), returning the name, kind and size for each file found (OE0528). The test case verifies that the wildcard patterns, "*" and "?", will only be applied following (to the right of) the farthest right-most forward-slash character in the input pattern parameter (OE0736). The test case also verifies that the *FileManager*'s *list* operation will return a zero length sequence if the search pattern is not found (OE0755). In addition, the test case must verify that the mounted *FileSystem* name is removed from the pathname of the input pattern parameter in the conditional case that the *FileManager* delegates the list operation to the mounted *FileSystem*(s) (OE0583).

Places to Verify

FileSystem, FileManager

IDL References

Data

```
struct FileInformationType {     string name;
CF::FileSystem::FileType  kind;
unsigned long long size;
CF::Properties fileProperties;};
typedef sequence <FileInformationType> FileInformationSequence;
```

Operations

```
CF::FileSystem::FileInformationSequence  list( in string pattern ) raises
        (CF::FileException, CF::InvalidFileName);
```

Preconditions

- The source code files of the Core Framework are available.
- Implementation of a file manager.
    - Note: if there is no implementation of a file manager, then the requirement is not applicable.

Test Description

For *FileManager*s of the OE under examination, perform the following steps for the *list* operation:

A. Verify that the *FileManager* implements or delegates the *list* operation of the mounted *FileSystem* (OE0582).
   1. **Pass**: The *FileManager* implements its own or delegates the *list* operation of the mounted *FileSystem*.
   2. **Fail**: The *FileManager* does not implement or delegate the *list* operation of the mounted *FileSystem*.

B. Verify that the *list* operation used by the *FileManager* returns a FileInformationSequence for files that match the search pattern (OE0543).
   1. **Pass**: The *list* operation returns a FileInformationSequence consisting of files that matches the search pattern.
   2. **Fail**: The *list* operation does not return a FileInformationSequence consisting of files that match the search pattern.

C. Verify that the FileInformationSequence returned by the *FileManager's list* operation includes name, kind, and size information for each returned file or directory (OE0528).
   1. **Pass**: The FileInformationSequence contains the name, kind and size information for each file or directory.
   2. **Fail**: The FileInformationSequence does not include name, kind, and size information for each returned file or directory.

D. Verify that the *FileManager* removes the name of the mounted file system from input pathnames it passes to the *list* operation of the mounted *FileSystem* (OE0583).
      a. **NOTE**: A *FileManager* may implement its own *list* operation or delegate the operation to the *FileSystem*s it has mounted. Under the condition that the *FileManager* implements its own *list* operation without delegating the operation to underlying *FileSystems*, it does not need to remove the name of the mounted *FileSystem* from the input pathname.
   1. **Pass**: The *FileManager* removes the name of the mounted *FileSystem* from the input pathname when passing the pathname to the *list* operation of the mounted *FileSystem*.
   2. **Fail**: The *FileManager* does not remove the name of the mounted *FileSystem* from the input pathname when passing the pathname to the *list* operation of the mounted *FileSystem*.

E. Verify that the *FileManager's list* operation supports the "*" and "?" wildcard characters to match any sequence of characters and any single character, respectively (OE0542).
      a. **NOTE**: There is an implicit interpretation in the requirement (OE0542) that the "*" and the "?" wildcard characters must work in combination with each other. The "?" symbol is used to match replacements of a single character and the "*" matches any sequence of characters. For example, the search term "file?ame*" must be able to find all files/directories that match the sequence with any single character replacing the question mark and any number/type of characters replacing the "*". The tester must use enough test patterns of the wildcards to thoroughly test the requirement.
   1. **Pass**: The FileManager's *list* operation supports the "*" and "?" wildcard characters.
   2. **Fail**: The FileManager's list operation does not support the use of the "*" or "?" wildcard characters.

F. Verify that the *FileManager*'s *list* operation only allows the wildcards, "*" and "?", following (to the right of) the right-most forward-slash character (OE0736).

1. **Pass**: The wildcards, "*" and "?", are only allowed following the right-most forward-slash character.
2. **Fail**: The *list* operation allows the wildcards, "*" and "?", to be placed before(to the left of) the right-most forward-slash character.

G. Verify that the *list* operation used by the *FileManager* returns a zero length sequence when no files are found that match the search pattern. (OE0755).
   1. **Pass**: The *list* operation returns a zero length sequence when no files are found that match the search pattern.
   2. **Fail:** The *list* operation does not return a zero length sequence when no files are found that match the search pattern.

Manual Test Steps

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

OE_TC_031				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the *FileManager* implements or delegates the *list* operation of the mounted *FileSystem* (OE0582).				
1. Locate the source code for the FileManager implementation.	The source code for the FileManager exists.			
2. Verify in the source code that the FileManager implements or delegates the list operation of the mounted FileSystem.	**Pass:** The *FileManager* does implements or delegates the *list* operation of the mounted *FileSystem*. (OE0582) **Fail**: The *FileManager* does not implement or delegate the *list* operation of the mounted *FileSystem*. (OE0582)			
B. Verify that the *list* operation used by the *FileManager* returns a FileInformationSequence for files that match the search pattern (OE0543).				

OE_TC_031				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Examine the source code files and verify that *list* operation used by the *FileManager* returns a FileInformationSequence *matching* the search pattern used.	**Pass:** The *list* operation returns a FileInformationSequence consisting of files that matches the search pattern. (OE0543) **Fail:** The *list* operation does not return a FileInformationSequence consisting of files that match the search pattern. (OE0543)		**NOTE**: There is the implication in this test case that *FileManager* must use inheritance properly to implement or delegate the *list* operation. If the list operation is delegated to the mounted FileSystem(s), then the list operation of the FileSystem must be checked. CF::*FileSystem*::FileInformationSequence *list* (in string pattern) raises (CF::FileException, CF::InvalidFileName); struct FileInformationType { string name; CF::*FileSystem*::FileType kind; unsigned long long size; CF::Properties fileProperties; };	
C. Verify that the FileInformationSequence returned by the *FileManager*'s list operation includes name, kind, and size information for each returned file or directory (OE0528).				

4. Examine the source code files and verify that the FileInformationSequence contains information for kind when the *FileManager* uses the *list* operation.	**Pass**: The FileInformationSequence contains the kind information for each file or directory. (OE0528)
	
	Fail: The FileInformationSequence does not include the kind information for each retumed file or directory. (OE0528)
			enum FileType {
			PLAIN,
			DIRECTORY,
			FILE_SYSTEM
			};	

OE_TC_031				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Examine the source code files and verify that the FileInformationSequence contains information for size when the *FileManager* uses the *list* operation.	**Pass**: The FileInformationSequence contains the size information for each file or directory. (OE0528) **Fail**: The FileInformationSequence does not include the size information for each returned file or directory. (OE0528)			
6. Examine the source code files and verify that the FileInformationSequence contains information for name when the *FileManager* uses the *list* operation.	**Pass**: The FileInformationSequence contains the name information for each file or directory. (OE0528) **Fail**: The FileInformationSequence does not include the name information for each returned file or directory. (OE0528)			

D. Verify that the *FileManager* removes the name of the mounted file system from input pathnames it passes to the *list* operation of the mounted *FileSystem* (OE0583).

NOTE: A *FileManager* may implement its own *list* operation or delegate the operation to the *FileSystem* it has mounted. Under the condition that the *FileManager* implements its own *list* operation without delegating the operation to any underlying *FileSystems*, it does not need to remove the name of the mounted *FileSystem* from the input pathname.

OE_TC_031				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Verify in the source code that the *FileManager* removes the name of the mounted file system from input pathnames when using the *FileSystems list* operation.	**Pass**: The *FileManager* removes the name of the mounted *FileSystem* from the input pathname when passing the pathname to the *list* operation of the mounted *FileSystem*. (OE0583) **Fail**: The *FileManager* does not remove the name of the mounted *FileSystem* from the input pathname when passing the pathname to the *list* operation of the mounted *FileSystem*. (OE0583)			
E. Verify that the *FileManager*'s *list* operation supports the "*" and "?" wildcard characters to match any sequence of characters and any single character, respectively (OE0542). **NOTE: There is an implicit interpretation in the requirement (OE0542) that the "*" and the "?" wildcard characters must work in combination with each other. The "?" symbol is used to match replacements of a single character and the "*" matches any sequence of characters. For example, the search term "file?ame*" must be able to find all files/directories that match the sequence with any single character replacing the question mark and any number/type of characters replacing the "*". The tester must use enough test patterns of the wildcards to thoroughly test the requirement.**				
8. Examine the source code files and verify that the *list* operation supports using the "*" wildcard to display files.	1. **Pass**: The FileManager's list operation supports the "*" wildcard characters. 2. **Fail**: The FileManager's list operation does not support the use of the "*" wildcard characters. (OE0542)			

OE_TC_031				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
9. Examine the source code files and verify that the FileManager's *list* operation supports using the "?" wildcard to display files.	1. **Fail**: The *FileManager*'s *list* operation does not support the use of the "?" wildcard characters. (OE0542) 2. **Pass**: The *FileManager*'s *list* operation supports the "?" wildcard characters. (OE0542)			
F. Verify that the *FileManager*'s *list* operation only allows the wildcards, "*" and "?", following (to the right of) the right-most forward-slash character (OE0736).				
10. Verify in the source code, that the FileManager's use of the *list* operation only allows the "?" or "*" wildcard to the right of the right-most forward-slash character for the input search parameter (/).	1. **Fail**: The list operation allows the wildcards, "*" and "?", to be placed before (to the left of) the right-most forward-slash character. (OE0736) 2. **Pass**: The wildcards, "*" and "?", are only allowed following the right-most forward-slash character. (OE0736)		Example: /directory/*/dir?Name/file_name would be an illegal search sequence because the "*" search character is not at the right-most end of path. This is also the case with dir?Name wildcard existing in a part of the path that is before the right-most forward-slash.	
G. Verify that the *list* operation used by the *FileManager* returns a zero length sequence when no files are found that match the search pattern. (OE0755).				
11. Examine the source code files and verify that the *list* operation used by the *FileManager* returns a zero length sequence if there are no files matching the search pattern.	**1. Fail**: The *list* operation does not return a zero length sequence when no files are found that match the search pattern. (OE0755) 2. **Pass**: The *list* operation returns a zero length sequence when no files are found that match the search pattern. (OE0755)		To implement this test, one might search for all files in the system and then search for the name of a file that does not exist (e.g. <path>/non_existent_file)	
End of Test				

Test Recording Log – OE_TC_031										
Step1 FileManager Code located	**Step2** *FileManager* implements/ delegates *list* operation	**Step3** *List* returns Correct FileInformation Sequence	**Step4** Sequence contains correct kind	**Step5** Sequence contains correct size	**Step6** Sequence contains correct name	**Step7** Removes pathname	**Step8** Accepts "*" search pattern	**Step9** Accepts "?" search pattern	**Step10** Does not allow "*" and "?" beyond right most slash character	**Step11** Returns zero when pattern isn't found

Test Summary – OE_TC_031

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

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

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

OE0528_____

OE0542_____

OE0543 _____

OE0582 _____

OE0583 _____

OE0736_____

OE0755_____

Failed Items (Section/Step Number): _____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.2. OE_TC_040 - Exceptions from the Mounted FileSystem

Test Case Number: OE_TC_040

Mounted FileSystem

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0739	The file manager shall propagate exceptions raised by a mounted file system.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-89, Section 3.1.3.4.3.5.4; Pages 3-79 thru 3-84, Section 3.1.3.4.2

Test Objective

This test case verifies requirement OE0739. The objective of this test will determine if the FileManager interface propagates or re-raises the exceptions just as if the FileSystem were accessed directly. Since FileManager is a class that inherits from FileSystem, it is possible that some of the operations are implicitly inherited from FileSystem; for such operations the exceptions raised are per the FileSystem exceptions (and requirement OE0739 is considered fulfilled). Where the FileManager redefines operations, the exceptions should be propagated (either caught and then reraised, or left uncaught) in order to fulfill requirement OE0739. There are three exceptions inherited from the FileSystem that the FileManager must raise (InvalidFileName, UnknownFileSystemProperties, and FileException).

Places to Verify

FileManager interface

IDL References

Exceptions

exception InvalidFileName {CF::ErrorNumberType errorNumber; string msg;}
exception UnknownFileSystemProperties {CF::Properties invalidProperties;};
exception FileException {CF::ErrorNumberType errorNumber; string msg;}

Operations

void remove (in string fileName) raises (FileException, InvalidFileName);

void copy (in string sourceFileName, in string destinationFileName) raises (InvalidFileName, FileException);
boolean exists (in string fileName) raises (InvalidFileName);
FileInformationSequence list (in string pattern) raises (FileException, InvalidFileName);
File create (in string fileName) raises (InvalidFileName, FileException);
File open (in string fileName, in boolean read_Only) raises (InvalidFileName, FileException);
void mkdir (in string directoryName) raises (InvalidFileName, FileException);
void rmdir (in string directoryName) raises (InvalidFileName, FileException);
void query (inout Properties fileSystemProperties) raises (UnknownFileSystemProperties);

Preconditions

- Domain Profile test has passed
- All the Domain Profile files are available
- The source code files are available

Test Description

A. Identify the source code files. (OE0739)
B. Identify the source code files that implement the *remove* operation for the FileManager. (OE0739)
    1. Verify that the *FileManager::remove* operation propagates the FileException exception if it is raised by the mounted FileSystem's *FileSystem::remove* operation.
        a. **Pass**: The FileManager inherits the *remove* operation from FileSystem and does not redefine it.
        b. **Pass**: The FileManager overwrites the *remove* operation and propagates* the *FileException*.
        c. **Fail**: The FileManager overwrites the *remove* operation but does not propagate* the *FileException*.

Note: * propagate = catches and re-raises the same exception, or doesn't catch the exception

    2. Verify that the *FileManager::remove* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::remove* operation.
        a. **Pass**: The FileManager inherits the *remove* operation from FileSystem and does not redefine it.
        b. **Pass**: The FileManager overwrites the *remove* operation and propagates* the *InvalidFileName* exception.
        c. **Fail**: The FileManager overwrites the *remove* operation but does not propagate* the *InvalidFileName* exception.

C. Identify the source code files that implement the *copy* operation for the FileManager. (OE0739)
    1. Verify that the *FileManager::copy* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::copy* operation.

a. **Pass**: The FileManager inherits the *copy* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *copy* operation and propagates* the *InvalidFileName* exception.
c. **Fail**: The FileManager overwrites the *copy* operation but does not propagate* the *InvalidFileName* exception.

2. Verify that the *FileManager::copy* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::copy* operation.
   a. **Pass**: The FileManager inherits the *copy* operation from FileSystem and does not redefine it.
   b. **Pass**: The FileManager overwrites the *copy* operation and propagates* the *FileException* exception.
   c. **Fail**: The FileManager overwrites the *copy* operation but does not propagate* the *FileException* exception.

D. Identify the source code files that implement the *exists* operation for the FileManager. (OE0739)
   1. Verify that the *FileManager::exists* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::exists* operation.
      a. **Pass**: The FileManager inherits the *exists* operation from FileSystem and does not redefine it.
      b. **Pass**: The FileManager overwrites the *exists* operation and propagates* the *InvalidFileName* exception.
      c. **Fail**: The FileManager overwrites the *exists* operation but does not propagate* the *InvalidFileName* exception.

E. Identify the source code files that implement the *list* operation for the FileManager. (OE0739)
   1. Verify that the *FileManager::list* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::list* operation.
      a. **Pass**: The FileManager inherits the *list* operation from FileSystem and does not redefine it.
      b. **Pass**: The FileManager overwrites the *list* operation and propagates* the *FileException* exception.
      c. **Fail**: The FileManager overwrites the *list* operation but does not propagate* the *FileException* exception.
   2. Verify that the *FileManager::list* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::list* operation.
      a. **Pass**: The FileManager inherits the *list* operation from FileSystem and does not redefine it.
      b. **Pass**: The FileManager overwrites the *list* operation and propagates* the *InvalidFileName* exception.
      c. **Fail**: The FileManager overwrites the *list* operation but does not propagate* the *InvalidFileName* exception.

F. Identify the source code files that implement the *create* operation for the FileManager. (OE0739)
   1. Verify that the *FileManager::create* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::create* operation.
      a. **Pass**: The FileManager inherits the *create* operation from FileSystem and does not redefine it.
      b. **Pass**: The FileManager overwrites the *create* operation and propagates* the *InvalidFileName* exception.
      c. **Fail**: The FileManager overwrites the *create* operation but does not propagate* the *InvalidFileName* exception.
   2. Verify that the *FileManager::create* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::create* operation.

a. **Pass**: The FileManager inherits the *create* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *create* operation and propagates* the *FileException* exception.
c. **Fail**: The FileManager overwrites the *create* operation but does not propagate* the *FileException* exception.

G. Identify the source code files that implement the *open* operation for the FileManager. (OE0739)
1. Verify that the *FileManager::open* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::open* operation.
a. **Pass**: The FileManager inherits the *open* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *open* operation and propagates* the *InvalidFileName* exception.
c. **Fail**: The FileManager overwrites the *open* operation but does not propagate* the *InvalidFileName* exception.
2. Verify that the *FileManager::open* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::open* operation.
a. **Pass**: The FileManager inherits the *open* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *open* operation and propagates* the *FileException* exception.
c. **Fail**: The FileManager overwrites the *open* operation but does not propagate* the *FileException* exception.

H. Identify the source code files that implement the *mkdir* operation for the FileManager. (OE0739)
1. Verify that the *FileManager::mkdir* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::mkdir* operation.
a. **Pass**: The FileManager inherits the *mkdir* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *mkdir* operation and propagates* the *InvalidFileName* exception.
c. **Fail**: The FileManager overwrites the *mkdir* operation but does not propagate* the *InvalidFileName* exception.
2. Verify that the *FileManager::mkdir* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::mkdir* operation.
a. **Pass**: The FileManager inherits the *mkdir* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *mkdir* operation and propagates* the *FileException* exception.
c. **Fail**: The FileManager overwrites the *mkdir* operation but does not propagate* the *FileException* exception.

I. Identify the source code files that implement the *rmdir* operation for the FileManager. (OE0739)
1. Verify that the *FileManager::rmdir* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::rmdir* operation.
a. **Pass**: The FileManager inherits the *rmdir* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *rmdir* operation and propagates* the *InvalidFileName* exception.
c. **Fail**: The FileManager overwrites the *rmdir* operation but does not propagate* the *InvalidFileName* exception.
2. Verify that the *FileManager::rmdir* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::rmdir* operation.

a. **Pass**: The FileManager inherits the *rmdir* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *rmdir* operation and propagates* the *FileException* exception.
c. **Fail**: The FileManager overwrites the *rmdir* operation but does not propagate* the *FileException* exception.

J. Identify the source code files that implement the *query* operation for the FileManager. (OE0739)
1. Verify that the *FileManager::query* operation propagates the *UnknownFileSystemProperties* exception if it is raised by the mounted FileSystem's *FileSystem::query* operation.
a. **Pass**: The FileManager inherits the *query* operation from FileSystem and does not redefine it.
b. **Pass**: The FileManager overwrites the *query* operation and propagates* the *UnknownFileSystemProperties* exception.
c. **Fail**: The FileManager overwrites the *query* operation but does not propagate* the *UnknownFileSystemProperties* exception.

Manual Test Steps

Notes:

1. Test Result will include Pass, Fail, Untested, or N/A.
2. The Test Recording Log is intended to record data for each step that requires recording of data.
3. The OE software engineer can be of assistance in locating the source code for the FileSystem operations. A search engine such as grep or a development tool such as Visual C++ can be used to search all the source code to find the various operations. Do not depend on Microsoft explorer's search operation, as it has demonstrated that it will not find a string within a file.

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files. (OE0739)				
1. Determine the source code files.	Source code files are recorded using the Test Recording Log.		The name of the executable in the SPD file is the best clue for the name of the source code.	
B. Identify the source code files that implement the *remove* operation for the FileManager.				
2. Locate the *remove* operation in the source code for the FileManager.	*remove* operation found.			
B.1 Verify that the FileManager::remove operation propagates the FileException exception if it is raised by the mounted FileSystem's FileSystem:: remove operation (OE0739).				
3. Examine the source code and verify that the *FileManager::remove* operation propagates the FileException exception if it is raised by the mounted FileSystem's *FileSystem::remove* operation.	**Pass**: The FileManager inherits the *remove* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *remove* operation and propagates* the *FileException*. (OE0739) **Fail**: The FileManager overwrites the *remove* operation but does not propagate* the *FileException*. (OE0739)			

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
B.2 Verify that the *FileManager::remove* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::remove* operation (OE0739).				
4. Examine the source code and verify that the *FileManager::remove* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::copy* operation.	**Pass**: The FileManager inherits the *remove* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *remove* operation and propagates* the *InvalidFileName* exception. (OE0739) **Fail**: The FileManager overwrites the *remove* operation but does not propagate* the *InvalidFileName* exception. (OE0739)			
C. Identify the source code files that implement the *copy* operation for the FileManager.				
5. Locate the *copy* operation in the source code for the FileManager.	copy operation found.			
C.1 Verify that the *FileManager::copy* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::copy* operation (OE0739).				
6. Examine the source code and verify that the *FileManager::copy* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::copy* operation.	**Pass**: The FileManager inherits the *copy* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *copy* operation and propagates* the *InvalidFileName* exception. (OE0739) **Fail**: The FileManager overwrites the *copy* operation but does not propagate* the *InvalidFileName* exception. (OE0739)			

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
C.2 Verify that the *FileManager::copy* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::copy* operation (OE0739).				
7. Examine the source code and verify that the *FileManager::copy* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::copy* operation.	**Pass**: The FileManager inherits the *copy* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *copy* operation and propagates* the *FileException* exception. (OE0739) **Fail**: The FileManager overwrites the *copy* operation but does not propagate* the *FileException* exception. (OE0739)			
D. Identify the source code files that implement the *exists* operation for the FileManager.				
8. Locate the *exists* operation in the source code for the FileManager.	*exists* operation found.			
D.1 Verify that the *FileManager::exists* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::exists* operation (OE0739).				
9. Examine the source code and verify that the *FileManager::exists* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::exists* operation**.**	**Pass**: The FileManager inherits the *exists* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *exists* operation and propagates* the *InvalidFileName* exception. (OE0739) **Fail**: The FileManager overwrites the *exists* operation but does not propagate* the *InvalidFileName* exception. (OE0739)			

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
E. Identify the source code files that implement the *list* operation for the FileManager.				
10. Locate the *list* operation in the source code for the FileManager.	*list* operation found.			
E.1 Verify that the *FileManager::list* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::list* operation (OE0739).				
11. Examine the source code and verify that the *FileManager::list* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::list* operation.	**Pass**: The FileManager inherits the *list* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *list* operation and propagates* the *FileException* exception. (OE0739) **Fail**: The FileManager overwrites the *list* operation but does not propagate* the *FileException* exception. (OE0739)			
E.2 Verify that the *FileManager::list* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::list* operation (OE0739).				
12. Examine the source code and verify that the. *FileManager::list* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::list* operation.	**Pass**: The FileManager inherits the *list* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *list* operation and propagates* the *InvalidFileName* exception. (OE0739) **Fail**: The FileManager overwrites the *list* operation but does not propagate* the *InvalidFileName* exception. (OE0739)		An absolute pathname starts with a (forward slash) character.	

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
F. Identify the source code files that implement the *create* operation for the FileManager.				
13. Locate *create* operation in the source code for the FileManager.	*create* operation found.			
F.1 Verify that the *FileManager::create* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::create* operation (OE0739).				
14. Examine the source code and verify that the *FileManager::create* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::create* operation**.**	**Pass**: The FileManager inherits the *create* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *create* operation and propagates* the *InvalidFileName* exception. (OE0739) **Fail**: The FileManager overwrites the *create* operation but does not propagate* the *InvalidFileName* exception. (OE0739)		An absolute pathname starts with a (forward slash) character.	
F.2 Verify that the *FileManager::create* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::create* operation (OE0739).				
15. Examine the source code and verify that the *FileManager::create* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::create* operation.	**Pass**: The FileManager inherits the *create* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *create* operation and propagates* the *FileException* exception. (OE0739) **Fail**: The FileManager overwrites the *create* operation but does not propagate* the *FileException* exception. (OE0739)			

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
G. Identify the source code files that implement the *open* operation for the FileManager.				
16. Locate the *open* operation in the source code for the FileManager.	*open* operation found.			
G.1 Verify that the *FileManager::open* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::open* operation (OE0739).				
17. Examine the source code and verify that the *FileManager::open* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::open* operation.	**Pass**: The FileManager inherits the *open* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *open* operation and propagates* the *InvalidFileName* exception. (OE0739) **Fail**: The FileManager overwrites the *open* operation but does not propagate* the *InvalidFileName* exception. (OE0739)			
G.2 Verify that the *FileManager::open* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::open* operation (OE0739).				
18. Examine the source code and verify that the *FileManager::open* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::open* operation.	**Pass**: The FileManager inherits the *open* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *open* operation and propagates* the *FileException* exception. (OE0739) **Fail**: The FileManager overwrites the *open* operation but does not propagate* the *FileException* exception. (OE0739)			

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
H. Identify the source code files that implement the *mkdir* operation for the FileManager.				
19. Locate the *mkdir* operation in the source code for the FileManager.	*mkdir* operation found.			
H.1 Verify that the *FileManager::mkdir* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::mkdir* operation (OE0739).				
20. Examine the source code and verify the *FileManager::mkdir* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::mkdir* operation.	**Pass**: The FileManager inherits the *mkdir* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *mkdir* operation and propagates* the *InvalidFileName* exception. (OE0739) **Fail**: The FileManager overwrites the *mkdir* operation but does not propagate* the *InvalidFileName* exception. (OE0739)			
H.2 Verify that the *FileManager::mkdir* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::mkdir* operation (OE0739).				
21. Examine the source code and verify that the *FileManager::mkdir* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::mkdir* operation.	**Pass**: The FileManager inherits the *mkdir* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *mkdir* operation and propagates* the *FileException* exception. (OE0739) **Fail**: The FileManager overwrites the *mkdir* operation but does not propagate* the *FileException* exception. (OE0739)			

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
I. Identify the source code files that implement the *rmdir* operation for the FileManager.				
22. Locate the *rmdir* operation in the source code for the FileManager.	*rmdir* operation found.			
I.1 Verify that the *FileManager::rmdir* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::rmdir* operation (OE0739).				
23. Examine the source code and verify that the *FileManager::rmdir* operation propagates the *InvalidFileName* exception if it is raised by the mounted FileSystem's *FileSystem::rmdir* operation.	**Pass**: The FileManager inherits the *rmdir* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *rmdir* operation and propagates* the *InvalidFileName* exception. (OE0739) **Fail**: The FileManager overwrites the *rmdir* operation but does not propagate* the *InvalidFileName* exception. (OE0739)			
I.2 Verify that the *FileManager::rmdir* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::rmdir* operation (OE0739).				
24. Examine the source code and verify that the *FileManager::rmdir* operation propagates the *FileException* exception if it is raised by the mounted FileSystem's *FileSystem::rmdir* operation.	.**Pass**: The FileManager inherits the *rmdir* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *rmdir* operation and propagates* the *FileException* exception. (OE0739) **Fail**: The FileManager overwrites the *rmdir* operation but does not propagate* the *FileException* exception. (OE0739)			

OE_TC_040				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
J. Identify the source code files that implement the *query* operation for the FileManager.				
25. Locate the *query* operation in the source code for the FileManager.	*query* operation found.			
J.1 Verify that the *FileManager::query* operation propagates the *UnknownFileSystemProperties* exception if it is raised by the mounted FileSystem's *FileSystem::query* operation (OE0739).				
26. Examine the source code and verify that the *FileManager::query* operation propagates the *UnknownFileSystemProperties* exception if it is raised by the mounted FileSystem's *FileSystem::query* operation.	**Pass**: The FileManager inherits the *query* operation from FileSystem and does not redefine it. (OE0739) **Pass**: The FileManager overwrites the *query* operation and propagates* the *UnknownFileSystemProperties* exception. (OE0739) **Fail**: The FileManager overwrites the *query* operation but does not propagate* the *UnknownFileSystemProperties* exception. (OE0739)			
End of Test				

Test Recording Log OE_TC_040				
Step1 (source code files)				
Step2 (remove)	**Step3** (remove–FileException)	**Step4** (remove- InvalidFileName)		
Step5 (copy)	**Step6** (copy– InvalidFileName)	**Step7** (copy – FileException)		
Step8 (exists)	**Step9** (exists– InvalidFileName)			
Step10 (list)	**Step11** (list –FileException)	**Step12** (list-InvalidFileName)		
Step13 (create)	**Step14** (create-InvalidFileName)	**Step15** (create –FileException)		
Step16 (open)	**Step17** (open-InvalidFileName)	**Step18** (open–FileException)		
Step19 (mkdir)	**Step20** (mkdir -InvalidFileName)	**Step21** (mkdir–FileException)		
Step22 (rmdir)	**Step23** (rmdir-InvalidFileName)	**Step24** (rmdir–FileException)		
Step25 (query)	**Step26** (query- UnknownFileSystemProperties)			

Test Summary: OE_TC_040

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

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

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

OE0739

remove _____ list _____ mkdir _____

copy _____ create _____ rmdir _____

exists _____ open _____ query _____

Failed Items (Section/Step Number):

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.4.3. OE_TC_060 - FileSystem Filename Lengths

Test Case Number: OE_TC_060

FileSystem

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0002	The OE and related file systems shall support a filename length of 40 characters and a pathname length of 1024 characters.
OE0605	Valid individual filenames and directory names shall be 40 characters or less.
OE0605-C123	Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters.
OE0605-C124	The filenames "." ("dot") and ".." ("dot-dot") are invalid in the context of a file system.
OE0733	A valid pathname shall not exceed 1024 characters.
OE0733-C125	Valid pathnames are structured according to the POSIX specification whose valid characters include the "/" (forward slash) character in addition to the valid filename characters.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-1, Section 3.1.1 Page 3-79 thru 3-86, Section 3.1.3.4.2.1

Test Objective

This test case verifies OE0002, OE0605, OE0733, OE0605-C123, OE0605-C124, and OE0733-C125. The objective of this test is to verify that the file systems of the OE will support a filename length of 40 characters, a path length of 1024 characters, and use valid characters. In this test case filename means both directory name and filename.

Places to verify

FileSystems

IDL References

Operations

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

```
void mkdir ( in string directoryName )
        raises (CF::InvalidFileName, CF::FileException);
```

Preconditions

- The CF file system source code files are available.

Test Description

Note: The file systems tested when validating OE0605 and OE0733 are the related file systems in requirement OE0002. If either of these requirements fails, then OE0002 fails.

For every file system implementation in the CF:

A. Locate the source code for the *create* operation of the file system. (OE0002, OE0605, OE0733)
   1. **Untested**: No source code can be found.
B. Verify that the *create* operation does not restrict filename length to less than 40 characters. (OE0002, OE0605)
   1. **Pass**: The filename is not restricted to less than 40 characters.
   2. **Fail**: The filename is restricted to less than 40 characters.
C. Verify that *create* operation allows only valid characters. (OE0605-C123)
   1. **Pass**: The *create* operation does not reject valid characters.
   2. **Fail** The *create* operation rejects valid characters.
D. Verify that the *create* operation rejects "." and ".." as file names. (OE0605-C124)
   1. **Pass**: The *create* operation rejects "." and ".." as file names.
   2. **Fail** The *create* operation does not reject "." and ".." as file names.
E. Verify that each *create* operation allows a pathname length of 1024 characters.(OE0002)
   1. **Pass**: The pathname is not restricted to less than 1024 characters.
   2. **Fail**: The pathname is restricted to less than 1024 characters.
F. Verify that each *create* operation restricts the pathname length to less than 1025 characters.(OE0733)
   1. **Pass**: The pathname is restricted to less than 1025 characters.
   2. **Fail**: The pathname is not restricted to less than 1025 characters.
G. Verify that the *create* operation accepts "/" (forward slash) as the file/directory name delimiter in the pathname. (OE0733-C125)
   1. **Pass**: The *create* operation accepts "/"as the file/directory name delimiter in the pathname.
   2. **Fail** The *create* operation does not accept "/"as the file/directory name delimiter in the pathname.
H. Locate the source code for the *mkdir* operation of the file system. (OE0002, OE0605, OE0733)
   1. **Untested**: No source code can be found.

I. Repeat steps B thru G for the *mkdir* operation. (OE0002, OE0605, OE0733, OE0605-C123, OE0605-C124, OE0733-C125)

Manual Test Steps

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

OE_TC_060				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the create operation of the file system. (OE0002, OE0605, OE0733)				
1. Locate the code for the *create* operation	**Untested**: No code for the *create* operation is found. (OE0002, OE0605, OE0733)			
B. Verify that the file system does not restrict filename length to less than 40 characters. (OE0002, OE0605)				
2. Verify that the *create* operation will not reject names that are 40 characters or less.	**Pass**: Names that are 40 characters or less are accepted. (OE0002, OE0605) **Pass**: Names that are greater than 40 characters are rejected. (OE0002, OE0605) **Fail**: Names that are 40 characters or less are rejected. (OE0002, OE0605) **Fail**: Names that are greater than 40 characters are accepted. (OE0002, OE0605)		Note that the underlying O/S may not allow names of 40 characters, but the *create* operation must accept them and compensate for the O/S deficiency.	
C. Verify that *create* operation allows only valid characters (OE0605-C123)				

OE_TC_060				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *create* operation will not reject valid characters.	**Pass**: Valid characters are not rejected. (OE0605-C123 **Fail**: Valid characters are rejected (OE0605-C123) (related requirements OE0002 and OE0605 fail)		Valid characters are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) and the "." (period), "_" (underscore) and "-" (hyphen) characters. Failure of this criterion OE0605-C123 means failure of the requirements OE0605.	
4. Verify that the *create* operation will reject invalid characters	**Pass**: Invalid characters are rejected. (OE0605-C123 **Fail**: Invalid characters are not rejected. (OE0605-C123) (related requirements OE0002 and OE0605 fail)		Failure of this criterion OE0605-C123 means failure of the requirements OE0605.	
D. Verify that the file system rejects "." and ".." as file names. (OE0605-C124)				
5. Verify that the *create* operation rejects "." and ".." as the filename (the rightmost name in a path).	**Pass**: "." and "..".are rejected. (OE0605-C124) **Fail**: : "." and "..".are not rejected. (OE0605-C124) (related requirements OE0002 and OE0605 fail)		Failure of this criterion OE0605-C124 means failure of the requirements OE0605.	
E. Verify that each file system allows a pathname length of 1024 characters. (OE0002)				

OE_TC_060				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Verify that the *create* operation allows the creation of a pathname that is 1024 characters or less.	**Pass**: Pathname length is not limited to less than 1024 characters. (OE0002) **Fail**: Pathname length of 1024 characters is rejected. (OE0002)		Note that the underlying O/S may not allow path lengths of 1024 characters, but the *create* operation must accept them and compensate for the O/S deficiency. In this case the following operations must be checked to verify that they compensate for the O/S deficiency: copy() exists() list() open() remove() rmdir()	
F. Verify that each file system restricts the pathname length to less than 1025 characters. (OE0733)				
7. Verify that the *create* operation limits the pathname length to less than 1025 characters.	**Pass**: Pathname length is limited to less than 1025 characters. (OE0733) **Fail**: Pathname length greater than 1024 characters is allowed. (OE0733)		The underlying O/S may allow greater pathname lengths, but the *create* operation must limit them to 1024 or less.	
G. Verify that the file system accepts "/" (forward slash) as the file/directory name delimiter in the pathname. (OE0733-C125)				
8. Verify that the *create* operation accepts "/" in the pathname.	**Pass**: "/" is accepted in the pathname. (OE0733-C125) **Fail**: "/" is not accepted in the pathname (OE0733-C125) (related requirements OE0002 and OE0733 fail)		If the underlying O/S does not accept "/" in the pathname, then the operation must transform "/" into a character that the O/S does accept. Failure of this criterion OE0733-C125 means failure of the requirements OE0733.	
H. Locate the source code for the *mkdir* operation of the file system. (OE0002, OE0605, OE0733)				

OE_TC_060				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
9. Locate the code for the *mkdir* operation	**Untested**: No code for the *mkdir* operation is found. (OE0002, OE0605, OE0733)			
I. Repeat steps B thru G for the *mkdir* operation. (Manual test steps 2 thru 8)				
Note: Steps 10 thru 16 repeat 2 thru 8 for the *mkdir* operation				
10. Verify that *mkdir* will not reject names that are 40 characters or less.	**Pass**: Names that are 40 characters or less are accepted. (OE0002, OE0605) **Pass**: Names that are greater than 40 characters are rejected. (OE0002, OE0605) **Fail**: Names that are 40 characters or less are rejected. (OE0002, OE0605) **Fail**: Names that are greater than 40 characters are accepted. (OE0002, OE0605)		Note that the underlying O/S may not allow names of 40 characters, but the *mkdir* operation must accept them and compensate for the O/S deficiency.	
11. Verify that the *mkdir* operation will not reject valid characters.	**Pass**: Valid characters are not rejected. (OE0605-C123) **Fail**: Valid characters are rejected. (OE0605-C123) (related requirements OE0002 and OE0605 fail)		Valid characters are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) and the "." (period), "_" (underscore) and "-" (hyphen) characters. Failure of this criterion OE0605-C123 means failure of the requirements OE0605.	

OE_TC_060				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
12. Verify that the *create* operation will reject invalid characters	**Pass**: Invalid characters are rejected. (OE0605-C123) **Fail**: Invalid characters are not rejected. (OE0605-C123) (related requirements OE0002 and OE0605 fail)		Failure of this criterion OE0605-C123 means failure of the requirements OE0605.	
13. Verify that the *mkdir* operation rejects "." and ".."as a directory name (the rightmost name in a path).	**Pass**: "." and ".." are rejected. (OE0605-C124) **Fail**: "." and ".." are not rejected. (OE0605-C124)(related requirements OE0002 and OE0605 fail)		Failure of this criterion OE0605-C124 means failure of the requirements OE0605.	
14. Verify that the *mkdir* operation allows the creation of a pathname that is 1024 characters or less.	**Pass**: Pathname length is not limited to less than 1024 characters. (OE0002) **Fail**: Pathname length of 1024 characters is rejected. (OE0002)		Note that the underlying O/S may not allow path lengths of 1024 characters, but the *create* operation must accept them and compensate for the O/S deficiency. In this case the following operations must be checked to verify that they compensate for the O/S deficiency: copy() exists() list() open() remove() rmdir()	
15. Verify that the *mkdir* operation limits the pathname length to less than 1025 characters.	**Pass**: Pathname length is limited to less than 1025 characters. (OE0733) **Fail**: Pathname length greater than 1024 characters is allowed. (OE0733)		The underlying O/S may allow greater pathname lengths, but the *mkdir* operation must limit them to 1024 or less.	

OE_TC_060				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
16. Verify that the *mkdir* operation accepts "/" in the pathname.	**Pass**: "/" is accepted in the pathname. (OE0733-C125) **Fail**: "/" is not accepted in the pathname. (OE0733-C125) (related requirements OE0002 and OE0733 fail)		If the underlying O/S does not accept "/" in the pathname, then the operation must transform "/" into a character that the O/S does accept. Failure of this criterion OE0733-C125 means failure of the requirements OE0733.	
End of Test				

Test Recording Log – OE_TC_060							
Step1/9 (source code)	**Step2/10** name length	**Step3/11** valid chars.	**Step4/12** invalid chars.	**Step5/13** rejects "." & ".."	**Step6/14** path allows 1024	**Step7/15** path limit	**Step8/16** "/"

Test Summary OE_TC_060

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

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

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

OE0002_____

OE0605_____

OE0605-C123_____

OE0605-C124_____

OE0733_____

OE0733-C125_____

Failed Items (Section/Step Number):

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.4. OE_TC_066 - File :: read raises IOException

Test Case Number: OE_TC_066

CF::File::read raises IOException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0515	The *read* operation shall raise the *IOException* when a *read* error occurs.
OE0507	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-77, Section 3.1.3.4.1.5.1.5

Test Objective

This test case verifies OE0515 and OE0507. The test will verify that when the File *read* operation through the Core Framework is executed that it throws an IOException exception when a read error occurs. In addition, the test will verify that the exception contains an error number of the type CF::ErrorNumberType.

Places to Verify

Source Code Files with File *read* operation

IDL References

Data

enum ErrorNumberType { CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP , CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE , CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT , CF_EXDEV };

Exceptions

exception IOException { ErrorNumberType errorNumber; string msg; };

Operations

void *read* (out CF::OctetSequence data,
in unsigned long length)
raises (CF::File::IOException);

Preconditions

- All of the source code files for the OE File component under test are available.

Test Description

A. Verify that the source code for the *File::read* operation is found. (OE0515)
   1. **Pass**: The source code for the *File::read* operation is found.
   2. **Untested:** The File component files are not found or identified.
   3. **Fail:** The source code for the *File::read* operation is not found.

For each File::*read* operation found within the OE under test do the following:

B. Verify that if a read error occurs, the *IOException* exception will be raised. (OE0515)
   1. **Pass:** The *IOException* exception is raised for an occurrence of a read error.
   2. **Fail:** The *IOException* exception is not raised for an occurrence of a read error.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0507)
   1. **Pass:** The *IOException* exception provides an error number of *ErrorNumberType* for an exception.
   2. **Fail:** The *IOException* exception does not provide an error number of *ErrorNumberType* for a file exception

Semi-automated Test Steps

After running JTAP's File write IOException test, perform manual step 4 for OE0507.

Manual Test Steps

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

OE_TC_066				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the source code for the *File::read* operation is found. (OE0515)				
1. Identify the source code for the File component.	**Pass:** The source file of the File component is found. (OE0515) **Untested:** The source file of the File component is not found. (OE0515)			
2. Locate and record the path name of the source code for the File::*read* operation of the File component.	**Pass**: Source file for the File::*read* operation is found. (OE0515) **Fail**: Source file for the File::*read* operation is not found. (OE0515)		void *read*(out CF::OctetSequence data, in unsigned long length) raises (CF::File::*IOException*);	
For each File::*read* operation found within the OE under test do the following:				
B. Verify that if a read error occurs, the *IOException* exception will be raised. (OE0515)				
3. Verify in the *read* operation code that if a read error occurs, then the *IOException* exception will be raised.	**Pass:** A read error causes the raising of the *IOException* exception. (OE0515) **Fail**: A read error does not cause the raising of any exception. (OE0515) **Fail**: A read error causes the raising of an exception other than the *IOException* exception. (OE0515)			

OE_TC_066				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0507)				
4. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0507) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0507)			
End of Test				

Test Recording Log – OE_TC_066		
Step 1 (Source Code for File component)		
Step 2 (Source Code for File::read operation)		
Step 3 (read error raises exception) Y/N	**Step 4** **(**error number in exception) Y/N	Notes

Test Summary OE_TC_066

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

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

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

OE0515_____

OE0507_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.5. OE_TC_067 - File :: close

Test Case Number: OE_TC_067

File::close

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0522	The *close* operation shall release any OE file resources associated with the component.
OE0523	The *close* operation shall make the file unavailable to the component.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-78, Section 3.1.3.4.1.5.4.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-10

Test Objective

This test case verifies OE0522 and OE0523. The test will verify that when the *close* operation through the CF::File interface is executed that it releases any OE file resources associated with the component. Furthermore, this test will verify that the *close* operation makes the particular file object unavailable to the component.

Places to Verify

File

IDL References

Operations

```
void close ()
        raises (CF::FileException);
```

Preconditions

- The File source code files are available.

Test Description

A. Identify the source code files that implement the File::*close* operation. (OE0522, OE0523)
   1. **Pass**: The files exist.
   2. **Fail**: The files are not found.

B. Verify that when the File::*close* operation is called on a file, that that file object is released. (OE0522, OE0523).
   1. **Pass**: There is call to File::*close* and the file object is released.
   2. **Fail**: There is call to File::*close* and the file object is not released.

Manual Test Steps

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

OE_TC_067				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *File::close* operation (OE0522, OE0523)				
1. Locate and record the source code files containing the File::*close* operation.	**Pass:** The files exist and are recorded. (OE0522, OE0523) **Fail:** The files for File::close() are not found, that is the files location is not determined. (OE0522, OE0523)		May need the help of the software engineer to locate the source code files.	
B. Verify that when the File::close operation is called on a file that that file object is released **(OE0522, OE0523).**				
2. Verify that the close operation releases the OE file resources associated with the component.	**Pass**: The *close* operation releases the OE file resources associated with the component. (OE0522) **Fail**: The *close* operation does not release the OE file resources associated with the component (OE0522)		OE file resources are anything that is managed by the CF::File class.	
3. Verify that the close operation makes the file unavailable to the component by releasing the object reference.	**Pass**: The *close* operation makes the file unavailable to the component by releasing the object reference. (OE0523) **Fail**: The *close* operation does not make the file unavailable to the component by releasing the object reference. (OE0523)		Making a file unavailable should include steps like releasing memory for the file record and assigning NIL to the file pointer. Other words that describe the outcome of the file are that it is *unreachable* and *unusable*.	
End of Test				

Test Recording Log – OE_TC_067		
Step 1 (Source code files)	**Step 2** (release of file resources) Y/N	**Step 3** (release of file object pointer) Y/N

Test Summary OE_TC_067

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

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

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

OE0522_____

OE0523_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.6. OE_TC_068 - File :: close raises FileException

Test Case Number: OE_TC_068

File::close raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0524	The *close* operation shall raise the CF *FileException* when it cannot successfully close the file.
OE0598	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-78, Section 3.1.3.4.1.5.4.5;
SCA Appendix B: AEP	Version 2.2.2 15 May 2006	all
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 through C-4, Page C-9, Section C.1

Test Objective

This test case verifies OE0524 and OE0598. The test will verify that the File::*close* operation raises the CF::FileException error when it cannot successfully close a file. It also verifies that the exception contains the error number.

Places to Verify

Core Framework File interface(s)

IDL References

Data

enum ErrorNumberType { CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP , CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE , CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT , CF_EXDEV };

Exceptions

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

Operations

void close () raises (CF::FileException);

Preconditions

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

Test Description

A. Identify the source code files that implement the *close* operation in the Core Framework. (OE0524)
   1. **Pass**: Source code files containing the *close* operation exist in the Core Framework.
   2. **Untested**: Source code files containing the *close* operation do not exist in the Core Framework.

B. Identify the implementation of the CF::FileException in the Core Framework. (OE0524)
   1. **Pass**: The implementation of CF::FileException exists in the Core Framework.
   2. **Fail**: The implementation of the CF::FileException does not exist in the Core Framework.

For every implementation of the *close* operation in the Core Framework, perform the following step(s):

C. Verify that the *close* operation raises the CF::FileException when a close error occurs. (OE0524)
   1. **Pass**: The *close* operation raises the CF::FileException when a close error occurs.
   2. **Fail**: The *close* operation does *not* raise the CF::FileException when a close error occurs.

D. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0598)
   1. **Pass:** The *fileException* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The *fileException* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Semi-automated Test Steps

After running JTAP's File sizeOf FileException, FileManager copy FileException, FileManager create FileException, FileManager list FileException, FileManager mkdir FileException, FileManager open FileException, FileManager remove FileException, and FileManager rmdir FileEx tests, perform manual step 4.

Manual Test Steps

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

OE_TC_068				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *close* operation in the Core Framework. (OE0524)				
1. Locate and record the source code files containing implementations of the *close* operation in the Core Framework.	**Pass:** Source code files implementing the *close* operation are located and recorded. (OE0524) **Untested**: Source code files implementing the *close* operation do not exist. (OE0524)		May need the help of the software engineer to locate the source code files. Note: The Core Framework may include the CF::File interface as well as separate File interfaces for different Operating Systems it is necessary for it to be compatible with.	
B. Identify the implementation of the CF::FileException in the Core Framework. (OE0524)				
2. Locate and record the files that implement the CF::FileException in the Core Framework.	**Pass:** Source code files containing the CF::FileException are located and recorded. (OE0524) **Fail**: Source code files containing the CF::FileException are not located. (OE0524)		This may include a source code implementation of the CF::FileException or the use of generated source code from SCA defined IDL.	
For every implementation of the *close* operation in the Core Framework, perform the following steps.				
C. Verify that the *close* operation raises the CF::FileException when a close error occurs. (OE0524)				

OE_TC_068				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Examine the source code files and verify that each *close* operation raises a CF::FileException when a close error occurs.	**Pass:** The *close* operation raises a CF::FileException when a close error occurs. (OE0524) **Fail:** The *close* operation does not raise a CF::FileException when a close error occurs. (OE0524)		Sample IDL code: void *close*() raises (CF::FileException);	
D. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0598)				
4. Determine if the exception includes an error number (errorNumber) of the type; ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0598)			
End of Test				

Test Recording Log – OE_TC_068			
Step 1 (Source code files of *close* operation)	**Step 2** (Source code files of CF::FileException) (Pass/Fail?)	**Step 3** (CF::FileException is raised when *close* operation encounters a file error?) (Pass/Fail?)	**Step 4** (CF::FileException contains error number of CF::ErrorNumberType) (Pass/Fail?)

Test Summary OE_TC_068

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

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

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

OE0524 _____

OE0598 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.7. OE_TC_069 - File :: setFilePointer raises FileException

Test Case Number: OE_TC_069

File::setFilePointer raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0526	The *setFilePointer* operation shall raise the CF *FileException* when the file pointer for the referenced file cannot be set to the value of the input filePointer parameter.
OE0598	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-78, Section 3.1.3.4.1.5.5.5

Test Objective

This test case verifies OE0526 and OE0598. The objective of this test is to verify that when the *setFilePointer* operation is executed, it raises the CF::*FileException* when the file pointer for the referenced file cannot be set to the value of the input *filePointer* parameter. The SCA states, "The *setFilePointer* operation positions the file pointer where the next read or write will occur". The *filePointer* attribute of the *setFilePointer* operation gets set with an input value.

Places to Verify

CF::File

IDL References

Data

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

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

Exceptions

FileException {
CF::ErrorNumberType errorNumber;
String msg;};

Operations

void setFilePointer (in unsigned long filePointer)
raises (CF::File::InvalidFilePointer, CF::FileException);

Preconditions

- All of the source code files with the *setFilePointer* operation for the OE under test are available.

Test Description

A. Identify the source code files that implement the File*::setFilePointer* operation. (OE0526)
   1. **Untested**: The source code files are not available.

For each File::*setFilePointer* operation found within the OE under test:

B. Verify that the File::*setFilePointer* operation raises the CF::*FileException* when the file pointer (the next read or write position) for the referenced file cannot be set to the value of the input *filePointer* parameter. (OE0526)
   1. **Fail**: The *setFilePointer* operation does not raise *FileException*.
   2. **Pass**: The *setFilePointer* operation raises *FileException*.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0598)
   1. **Pass:** The *fileException* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The *fileException* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Semi-automated Test Steps

After running JTAP's File sizeOf FileException, FileManager copy FileException, FileManager create FileException, FileManager list FileException, FileManager mkdir FileException, FileManager open FileException, FileManager remove FileException, and FileManager rmdir FileEx tests, perform manual step 5.

Manual Test Steps

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

OE_TC_069				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the File *setFilePointer* operation.				
1. Perform a keyword search for *setFilePointer* operation on all source code provided by the developer.	The directories are recorded. **Untested:** No results from keyword search. (OE0526)		May need the help of the software engineer to locate the source code files directories.	
2. Examine the source code files returned in Step 1 and search for setFilePointer operation implementation. Record the file name.	setFilePointer operation for each implementation is identified and recorded.			
B. Verify that the File setFilePointer operation raises the CF::FileException when the file pointer (the next read or write position) for the referenced file cannot be set to the value of the input filePointer parameter. (OE0526)				
3. Search within the setFilePointer operation where the file pointer is being set to the value of the input filePointer parameter.	Identified the source code where setFilePointer sets the next read or write position.			
4. Determine that *CF::FileException* is raised when the file pointer cannot be set.	**Pass:** The *setFilePointer* operation raises the CF::*FileException*. (OE0526) **Fail:** The *setFilePointer* operation does not raise the			

OE_TC_069				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
	CF::*FileException*. (OE0526)			
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0598)				
5. Determine if the exception includes an error number (ErrorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (ErrorNumber) of ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (ErrorNumber) of ErrorNumberType. (OE0598)			
End of Test				

Test Recording Log – OE_TC_069				
Step 1 (Directory name location)	Step 2 (Source file name)	Step 4 (FileException raised)	Step 5 (error value is ErrorNumberType)	Notes

Test Summary OE_TC_069

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

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

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

OE0526_____

OE0598_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.8. OE_TC_091 - Framework Services Interfaces

Test Case Number: OE_TC_091

Framework Services Interfaces

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0506	Framework Services Interfaces shall be implemented using the CF IDL presented in Appendix C.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-75, Section 3.1.3.4 Appendix C (CF IDL)

Test Objective

This test case verifies requirement OE0506. The objective of this test is to see whether the CF IDL and the CF source code are a match for the CF IDL that is in the SCA v2.2.2 Appendix C.

Places to Verify

File, FileSystem, FileManager

IDL References

None.

Preconditions

- The CF IDL files are available.
- Language-specific (e.g. C++, Ada, Java) source code that has been compiled from the CF IDL in SCA Appendix C.

Test Description

A. Verify that the OE IDL used is correct by comparing against CF IDL in SCA Appendix C.
   1. Verify that the File interface in the OE IDL matches the CF IDL File interface. (OE0506)
      a. **Pass:** File interface matches

   b. **Fail:** File interface doesn't match
2. Verify that the FileSystem interface in the OE IDL matches the CF IDL FileSystem interface. (OE0506)
   a. **Pass:** FileSystem interface matches
   b. **Fail:** FileSystem interface doesn't match
3. Verify that the FileManager interface in the OE IDL matches the CF IDL FileManager interface. (OE0506)
   a. **Pass:** FileManager interface matches
   b. **Fail:** FileManager interface doesn't match

B. Verify that the implementation source files are derived from the CF IDL classes.
1. Verify that the File class in the OE source code is inherited from the CF IDL classes. (OE0506)
   a. **Pass:** File class matches
   b. **Fail:** File class doesn't match
2. Verify that the FileSystem class in the OE source code is inherited from the CF IDL classes. (OE0506)
   a. **Pass:** FileSystem class matches
   b. **Fail:** FileSystem class doesn't match
3. Verify that the FileManager class in the OE source code is inherited from the CF IDL classes. (OE0506)
   a. **Pass:** FileManager class matches
   b. **Fail:** FileManager class doesn't match

Manual Test Steps

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

OE_TC_091				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the CF IDL used is correct by comparing against CF IDL in SCA Appendix C (OE0506).				
1. Verify that the File interface in the OE IDL matches the CF IDL File interface. This comparison can be performed using a file comparison tool (e.g. windiff, fc, codewrite).	**Pass**: File interface matches (OE0506) **Fail**: File interface does not match. (OE0506)			
2. Verify that the FileSystem interface in the OE IDL matches the CF IDL FileSystem interface	**Pass**: FileSystem interface matches (OE0506) **Fail**: FileSystem interface does not match. (OE0506)			
3. Verify that the FileManager interface in the OE IDL matches the CF IDL FileManager interface	**Pass**: FileManager interface matches (OE0506) **Fail**: FileManager interface does not match. (OE0506)			
B. Verify that the implementation source files are derived from the CF IDL classes (OE0506).				
4. Verify that the File class in the OE source code is inherited from the CF IDL classes.	**Pass**: File class matches (OE0506) **Fail**: File class does not match. (OE0506)			
5. Verify that the FileSystem class in the OE source code is inherited from the CF IDL classes.	**Pass**: FileSystem class matches (OE0506) **Fail**: FileSystem class does not match. (OE0506)			

OE_TC_091				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Verify that the FileManager class in the OE source code is inherited from the CF IDL classes.	**Pass**: FileManager class matches. (OE0506) **Fail**: FileManager class does not match. (OE0506)			
End of Test				

Test Recording Log – OE_TC_091					
Step 1 OE File IDL matches the CF IDL File interface	**Step 2** OE FileSystem IDL matches the CF IDL FileSystem interface	**Step 3** OE FileManager IDL matches the CF IDL FileManager interface	**Step 4** OE File class inherits from CF IDL File classes	**Step 5** OE FileSystem class inherits from CF IDL FileSystem classes	**Step 6** OE FileManager class inherits from CF IDL FileManager classes

Test Summary OE_TC_091

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

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

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

OE0506 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.9. OE_TC_103 - FileSystem :: list raises InvalidFileName

Test Case Number: OE_TC_103

CF::FileSystem::list raises InvalidFileName
CF::FileManager::list raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0545	The *list* operation shall raise the CF InvalidFileName exception when the input pattern parameter is not an absolute pathname or cannot be interpreted due to unexpected characters.
OE0599	The CF InvalidFileName exception indicates an invalid file name was passed to a file service operation. The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-83, Section 3.1.3.4.2.5.4.5; Page 3-93, Section 3.1.3.6.4; Page 1-2, Section 1.3.1.1; Page 3-79, Section 3.1.3.4.2.1
SCA Appendix C: Core Framework IDL		Page C-3 through C-4, Section C-1; Page C-7, Section C-1.

Test Objective

This test case verifies OE0545 and OE0599. The objective of this test is to verify that the *list* operation raises the CF::InvalidFileName exception when the input pattern parameter is not an absolute pathname or cannot be interpreted due to unexpected characters. The requirement, OE0599, verifies that the error number contains one of the values listed in the CF::ErrorNumberType enumeration.

Places to Verify

File systems and file managers of the Core Framework.

IDL References

Data

enum ErrorNumberType {

CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP ,CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE , CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT ,CF_EXDEV };

Exceptions

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

Operations

CF::FileSystem::FileInformationSequence list (in string pattern)
raises (CF::FileException, CF::InvalidFileName);

Preconditions

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

Test Description

A. Identify the implementations of the *list* operation in the Core Framework. (OE0545)
Note: There may be several implementations of the *list* operation in the file managers or file systems of the CF; Some of these implementations may inherit from each other.
1. **Pass:** The implementations of the *list* operation are identified in the Core Framework.
2. **Untested**: The implementations of the *list* operation are not identified in the Core Framework.

B. Verify that CF::InvalidFileName exception exists in the Core Framework. (OE0545)
1. **Pass**: The CF::InvalidFileName exception exists in the Core Framework.
2. **Fail**: The CF::InvalidFileName exception does not exist in the Core Framework.

For each implementation of the list operation in the Core Framework, perform all of the following steps:

C. Verify that the *list* operation raises the CF::InvalidFileName exception when the input pattern parameter is not an absolute pathname. (OE0545)
1. **Pass**: The *list* operation raises the CF::InvalidFileName exception when the input pattern parameter is not an absolute pathname.
2. **Fail:** The *list* operation does *not* raise the CF::InvalidFileName exception when the input pattern parameter is not an absolute pathname.
3. **Fail:** The *list* operation raises the CF::InvalidFileName exception when the input pattern parameter is an absolute pathname.

D. Verify that the *list* operation raises the CF::InvalidFileName exception when the input pattern parameter cannot be interpreted due to unexpected characters. (OE0545)
   1. **Pass**: The *list* operation raises the CF::InvalidFileName exception when the input pattern parameter cannot be interpreted due to unexpected characters.
   2. **Fail**: The *list* operation does not raise the CF::InvalidFileName exception when the input pattern parameter cannot be interpreted due to unexpected characters.
   3. **Fail**: The *list* operation raises the CF::InvalidFileName exception when the input pattern parameter contains valid characters and is a valid pathname that exists on the operating system.

E. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)
   1. **Pass:** The CF::InvalidFileName exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The CF::InvalidFileName exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_103				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the *list* operation in the Core Framework. (OE0545) **Note: there may be several implementations of the *list* operation in the file managers or file systems of the OE; Some of these implementations may inherit from each other.**				
1. Examine the source code of the Core Framework and identify all of the implementations of the *list* operation. List the source code files that implement the *list* operation.	**Pass**: The implementations of the *list* operation are identified in the CF source code. (OE0545) **Untested**: The implementations of the *list* operation are *not* identified in the CF source code. (OE0545)		In some cases, the functionality of one *list* operation may be inherited through another implementation of the *list* operation.	
B. Verify that CF::InvalidFileName exception exists in the Core Framework. (OE0545)				
2. Verify in the source code of the Core Framework that the CF::InvalidFileName exception exists.	**Pass:** The CF::InvalidFileName exception exists in the CF source code. (OE0545) **Fail:** The CF::InvalidFileName exception does not exist in the CF source code. (OE0545)			
For each implementation of the *list* operation in the Core Framework, perform all of the following steps:				
C. Verify that the *list* operation raises the CF::InvalidFileName exception when the input pattern parameter is not an absolute pathname. (OE0545)				

OE_TC_103				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify in the source code of the *list* operation that the CF::InvalidFileName exception is raised when the input pattern parameter does not start with a forward slash, which indicates that it is an absolute pathname.	**Pass**: The *list* operation raises the CF::InvalidFileName exception when the input parameter is not an absolute pathname. (OE0545) **Fail**: The *list* operation does not raise the CF::InvalidFileName exception when the input parameter is not an absolute pathname. (OE0545) **Fail**: The *list* operation raises the CF::InvalidFileName exception when the input parameter is an absolute pathname. (OE0545)		Section 1.3.1.1 page 1-2 "An "absolute pathname" is a pathname which starts with a (forward slash) character"	
D. Verify that the *list* operation raises the CF::InvalidFileName exception when the input pattern parameter cannot be interpreted due to unexpected characters. (OE0545)				
4. Verify in the source code that the *list* operation raises the CF::InvalidFileName exception when the input pattern parameter contains characters that are not considered by the SCA as being valid.	**Pass**: The CF::InvalidFileName exception is raised when characters are included that are not considered by SCA as being valid. (OE0545) **Fail**: The CF::InvalidFileName exception is not raised when the pathname includes characters that are not considered by SCA as being valid. (OE0545) **Fail**: The CF::InvalidFileName exception is raised when the pathname includes only valid characters. (OE0545)		Section 3.1.3.4.2.1 Page 3-79 "Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters. The filenames "." ("dot") and ".." ("dot-dot") are invalid in the context of a file system."	

OE_TC_103				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Verify in the source code that the *list* operation does not raise the CF::InvalidFileName exception when the input pattern contains either the "*" or "?" character after the last forward slash, indicating a search pattern.	**Pass**: The CF::InvalidFileName exception is not raised if the input pattern contains either the "*" or "?" character after the last forward slash. (OE0545) **Fail**: The CF::InvalidFileName exception is raised if the input pattern contains either the "*" or "?" character after the last forward slash. (OE0545)		The "*" and "?" are considered valid characters, and the InvalidFileName exception must not be raised as long as they are placed after the last forward slash.	
E. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)				
6. Determine if the exception includes an error number (errorNumber) of the type; ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_103					
Step1 Source code file of list operations(s)	**Step2** (CF::InvalidFileName exception exists?) (Y/N?)	**Step3** (exception raised when pattern isn't an absolute pathname?) (Y/N?)	**Step4** (exception raised when pattern contains invalid characters?) (Y/N?)	**Step5** (processes '*' and "?" with no exception?) (Y/N?)	**Step6** (error number of exception is a CF::ErrorNumberType value?) (Y/N?)

Test Summary OE_TC_103

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

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

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

OE0545 _____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.10. OE_TC_104 - FileSystem :: list raises FileException

Test Case Number: OE_TC_104

FileSystem::list raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0546	The list operation shall raise the CF FileException when a file-related error occurs.
OE0598	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-83, Section 3.1.3.4.2.5.4.5; Page 3-93, Section 3.1.3.6.3;
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-3, Section C.1

Test Objective

This test case verifies the requirement OE0546 and OE0598. The objective of this test is to verify that the *list* operation raises the CF::FileException when a file-related error occurs (OE0546). The error number of the exception must be one of the CF::ErrorNumberType values.

Places to Verify

File systems and file managers of the Core Framework(CF)

IDL References

Data

enum ErrorNumberType { CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP ,CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE , CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT ,CF_EXDEV };

Exceptions

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

Operations

CF::FileSystem::FileInformationSequence list (in string pattern)
raises (FileException, InvalidFileName) ;

Preconditions

- All of the source code files with the *list* operation for the CF under test are available.

Test Description

A. Identify the source code files that implement the *list* operation. (OE0546)
   1. **Untested**: The source code files of the *list* operation are not available.

For each implementation of the *list* operation in the Core Framework, perform all of the following steps:

B. Verify that the *list* operation raises the CF::FileException when a file related error occurs. (OE0546)
   1. **Pass**: The *list* operation raises the CF::FileException when retrieving the list of files incurs a file related error.
   2. **Fail:** The *list* operation does *not* raise the CF::FileException when retrieving the list of files incurs a file related error.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0598)
   1. **Pass:** The CF::*FileException* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The CF::*FileException* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_104				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the *list* operation in the Core Framework. (OE0546)				
1. Examine the source code of the OE and identify all of the implementation(s)(may be more than one) of the *list* operation in the OE.	**Untested**: No *list* operation implementations are identified in the OE. (OE0546)		There may be several implementations of the list operation in the file managers or file systems of the OE; some of these implementations may inherit from each other.	
For each implementation of the *list* operation in the Core Framework, perform all of the following steps:				
B. Verify that the *list* operation raises the CF::FileException when a file related error occurs. (OE0546)				
2. Verify in the source code that the *list* operation raises the CF::FileException when retrieving the list of files incurs a file related error.	**Pass**: The *list* operation raises the CF::FileException when retrieving the list of files incurs a file related error. (OE0546) **Fail**: The *list* operation does *not* raise the CF::FileException when retrieving the list of files incurs a file related error. (OE0546)		The proper functionality(throwing the CF::FileException) may be implemented within the *list* operation under examination or through inheritance of another *list* operation implementation.	
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0598)				

OE_TC_104				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Determine if the exception includes an error number (errorNumber) of the type CF::ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of CF::ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0598)			
End of Test				

Test Recording Log - OE_TC_104		
Step 1 (Source code file for the *list* operation)	**Step 2** (CF::FileException is raised when there is an error retrieving the list of files.) (Pass/Fail?)	**Step 3** (exception contains errorNumber of CF::ErrorNumberType.) (Pass/Fail?)

Test Summary OE_TC_104

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

Pass: No failures detected
Fail: Failure(s) detected in Step(s) (x). Failure of the associated criterion means that the requirement...
Untested: Condition which is not testable
N/A: Not Applicable

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

OE0546 _____

OE0598 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.11. OE_TC_105 - FileSystem :: remove raises FileException

Test Case Number: OE_TC_105

CF::FileSystem::remove raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0534	The *remove* operation **shall** raise the CF FileException when a file-related error occurs.
OE0598	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
SCA Specification Version 2.2.2 - *remove* operation Exception - *FileException*	Version 2.2.2 15 May 2006	 Page 3-81, Section 3.1.3.4.2.5.1.5 Page 3-93, Section 3.1.3.6.3

Test Objective

This test case verifies OE0534 and OE0598. The objective of this test case is to verify that the *remove* operation raises the CF::*FileException* when a file-related error occurs. This test case also verifies that the error number value in the exception is of the type CF::ErrorNumberType.

Places to Verify

FileSystem; FileManager

IDL References

Data

```
enum ErrorNumberType {
        CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
        CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS,
        CF_EINTR,CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG,
        CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS,
        CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP ,CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE ,
        CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT ,CF_EXDEV };
```

Exceptions

exception FileException {ErrorNumberType errorNumber; string msg; };

Operations

void remove (in string fileName)
raises (FileException, InvalidFileName);

Preconditions

- Source code files for the OE FileSystem under test are available.

Test Description

A. Verify that the source code for the *FileSystem::remove* operation is found. (OE0534)
   1. **Pass**: The source code for the *FileSystem::remove* operation is found.
   2. **Untested:** The FileSystem files are not found or identified.
   3. **Fail:** The source code for the *FileSystem::remove* operation is not found.

For each *FileSystem::remove* operation found within the OE under test:

B. Verify that the source code for the throwing of the *FileException* exception is found. (OE0534)
   1. **Pass:** The source code contains the throwing of the *FileException* exception.
   2. **Fail:** The source code does not contain the throwing of the *FileException* exception.

C. Verify that if a file related error occurs, the *FileException* exception will be raised. (OE0534)
   1. **Pass:** The *FileException* exception is raised for an occurrence of a file related error.
   2. **Fail:** The *FileException* exception is not raised for an occurrence of a file related error.

D. Verify that the error number that accompanies the exception is a CF ErrorNumberType value. (OE0598)
   1. **Pass:** The *FileException* exception provides an error number of *ErrorNumberType* for a file exception.
   2. **Fail:** The *FileException* exception does not provide an error number of *ErrorNumberType* for a file exception

Manual Test Steps

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

OE_TC_105				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the source code for the *FileSystem::remove* operation is found. (OE0534)				
1. Identify the source code for the FileSystem.	**Pass:** FileSystem is found. (OE0534) **Untested:** FileSystem is not found. (OE0534)			
2. Locate and record the full path name of the source code for the implementation of the *remove* operation of the FileSystem.	**Pass**: File is found. (OE0534) **Fail**: File is not found. (OE0534)		void remove(in string fileName) raises (FileException, InvalidFileName);	
B. Verify that the source code for the throwing of the *FileException* exception is found. (OE0534)				
3. Locate and record the full path name of the source code file and directory of the thrown *FileException* exception.	**Pass**: File is found.(OE0534) **Fail**: File is not found. (OE0534)		exception FileException { ErrorNumberType errorNumber; string msg; };	
C. Verify that if a file related error occurs, the *FileException* exception will be raised. (OE0534)				
4. Verify in the *remove* operation code that if a file related error occurs, then the *FileException* exception will be raised.	**Pass:** A file related error causes the raising of the *FileException* exception. (OE0534) **Fail**: A file related error occurs and the FileException exception is not raised. (OE0534)		Record the file errors that actually cause the FileException exception. The *remove* operations can raise 2 different exceptions. Make sure the file related errors raise the *FileException* exception.	
D. Verify that the error number that accompanies the exception is a CF ErrorNumberType value. (OE0598)				

OE_TC_105				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Determine if the exception includes an error number (errorNumber) of the type; ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0598)			
End of Test				

Test Recording Log – OE_TC_105					
Step 1 (Source Code for FileSystem)	**Step 2** (Source Code for Remove operation)	**Step 3** (Source Code for FileException)	**Step 4** (file error raises exception) Y/N	**Step 5** (error number in exception) Y/N	**Notes**

Test Summary OE_TC_105

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

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

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

OE0534 _____

OE0598 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.12. OE_TC_106 - FileSystem :: copy

Test Case Number: OE_TC_106

FileSystem::*copy*

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0535	The *copy* operation shall *copy* the source file identified by the input *sourceFileName* parameter to the destination file identified by the input *destinationFileName* parameter.
OE0734	The *copy* operation shall overwrite the destination file, when the destination file already exists and is not identical to the source file.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-82, Section 3.1.3.4.2.5.2.3

Test Objective

This test case verifies OE0535 and OE0734. The test is to verify that the *copy* operation found in the FileSystem and FileManager copies the file identified by the input parameter *sourceFileName* to the destination file identified by the input parameter *destinationFileName*. It also verifies that should the destination file exist and differ from the source file, the *copy* operation will overwrite it with the source file. The files mentioned in these requirements are plain files.

Places to Verify

FileSystem, FileManager

IDL References

Operations

void *copy* (in string sourceFileName,
in string destinationFileName) raises (InvalidFileName, FileException);

Preconditions

- Source code files with the *copy* operation for the FileSystem and FileManager Framework Services for the OE under test are available.

- The IDL interface for the *copy* operation for the FileSystem and FileManager Framework Services has been verified. This is requirement OE0506.

Test Description

For the *copy* operation within the FileSystem and FileManager of the OE under test:

A. Identify the source code for all *copy* operations within the FileSystem and FileManager. (OE0535, OE0734).
   1. **Pass**: Source code for the *copy* operation is found.
   2. **Untested:** Source code for the *copy* operation is not found.

For every file with a *copy* operation implemented, perform all of the following:

B. Verify that the *copy* operation copies the file identified by the input parameter called *sourceFileName* into the file identified by the input parameter *destinationFileName*. (OE0535, OE0734).
   1. Verify that the *copy* operation copies the contents of the file identified by the input parameter called *sourceFileName* into the file identified by the input parameter *destinationFileName*.(OE0535)
      a. **Pass**: The *copy* operation copies the file identified by the input parameter *sourceFileName* into the file identified by the input parameter *destinationFileName*.
      b. **Fail**: The *copy* operation does not *copy* the file identified by the input parameter *sourceFileName* into the file identified by the input parameter *destinationFileName*.
   2. Verify, if the *destinationFileName* **file** already exists and differs from the *sourceFileName* file, that the *copy* operation overwrites the existing file.(OE0734)
      a. **Pass**: The *copy* operation overwrites the *destinationFileName* file with the file identified by the input parameter *sourceFileName*.
      b. **Fail**: The *copy* operation does not overwrite the *destinationFileName* file with the file identified by the input parameter *sourceFileName*.

Manual Test Steps

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

OE_TC_106				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code for all *copy* operations within the FileSystem and FileManager. (OE0535, OE0734).				
1. Record the location and name of the source code for the *copy* operations, which conform to this requirement.	**Pass**: Files are found. (OE0535, OE0734). **Untested**: Files are not found. (OE0535, OE0734).			
For every file with a *copy* operation implemented, perform all of the following steps:				
B. Verify that the *copy* operation copies the contents of the file identified by the input parameter called *sourceFileName* into the file identified by the input parameter *destinationFileName*. (OE0535, OE0734).				
2. Verify, if the *destinationFileName* file already exists and differs from the *sourceFileName* file, that the *copy* operation overwrites the existing file.	**Pass:** The *copy* operation copies the file identified by *sourceFileName* into the file identified by *destinationFileName*. (OE0734) **Fail:** The *copy* operation does not *copy* the file identified by **sourceFileName** into the file identified by ***destinationFileName***. (OE0734)			

OE_TC_106				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *copy* operation copies the file identified by *sourceFileName* into a file identified by *destinationFileName*.	**Pass:** The *copy* operation copies the file identified by *sourceFileName* into the file identified by *destinationFileName*. (OE0535) **Fail:** The *copy* operation does not *copy* the file identified by *sourceFileName* into the file identified by *destinationFileName*. (OE0535)			
End of Test				

Test Recording Log – OE_TC_106					
Step 1 (*copy* operation Source Code**)**	**Step 2** (*copy* operation return type is void?) Y/N	**Step 3** (Two string input parameters?) Y/N	**Step 4** (Parameters properly named?) Y/N	**Step 5** (Overwrites if the destination file exists?) Y/N	**Step 6** (Source file is copied into destination file?) Y/N

Test Summary OE_TC_106

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

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

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

OE0535_____

OE0734_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.13. OE_TC_107 - FileSystem :: create

Test Case Number: OE_TC_107

FileSystem::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0547	The *create* operation shall create a new File based upon the input *fileName* parameter.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-83, Section 3.1.3.4.2.5.5.3

Test Objective

This test case verifies requirement OE0547. The objective of this test is to verify that a new file is created via the *create* operation using the input *fileName* parameter.

Places to Verify

FileManager

IDL References

Operations

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

Exceptions

exception InvalidFileName { CF::ErrorNumberType errorNumber; string msg; };
exception FileException { CF::ErrorNumberType errorNumber; string msg; };

Preconditions

- OE source code files having the implementation of the *create* operation are available.

Test Description

A. Identify the source code files that implement the *create* operation. (OE0547)

1. **Untested**: The source code files having the implementation of *create* are not available.

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

B. Verify that the *create* operation creates a new file based on the input *fileName* parameter. (OE0547)

3. **Pass**: The *create* operation creates a new file.
4. **Fail:** The *create* operation does not create a new file.

Manual Test Steps

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

OE_TC_107				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *create* operation. (OE0547)				
1. Perform a keyword search for *create* operation on all source code provided by the developer and record the source code directories.	The source code directories are recorded.		Multiple FileSystems may exist. May need the help of the software engineer to locate the source code files directories.	
2. Examine the source code files returned from Step 1 and search for *create* operation implementation. Record the file name.	The *create* operation for each implementation is identified and recorded. **Untested**: The source code files having the implementation of *create* are not available. **(OE0547)**			
For each *create* operation found within the OE under test, perform the following steps:				
B. Verify that the *create* operation creates a new file based on the input *fileName* parameter. (OE0547)				
3. Verify that a new file is created by the *create* operation based on the input *fileName* parameter.	**Pass**: The *create* operation creates a new file. **OE0547)** **Fail:** The *create* operation does not create a new file. **OE0547)**			
End of Test				

Test Recording Log – OE_TC_107		
Step 1 (source code directories)	**Step 2** (source code files)	**Step 3** (new file created-Y/N?)

Test Summary OE_TC_107

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

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

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

OE0547 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.4.14. OE_TC_109 - FileSystem's CREATED_TIME_ID property

Test Case Number: OE_TC_109

FileSystem

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0529	For this property, the identifier is CREATED_TIME_ID and the value shall be an unsigned long long data type containing the number of seconds since 00:00:00 UTC, Jan. 1, 1970.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-79, Figure 3-30; Page 3-81, Section 3.1.3.4.2.3.6

Test Objective

This test case verifies OE0529. The objective of this test is to verify that the CREATED_TIME_ID property is implemented and that the data type is correct.

Places to Verify

FileSystem

IDL References

Data

```
const string CREATED_TIME_ID = "CREATED_TIME";

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

Preconditions

- The Core Framework source code files are available.

Test Description

A. Locate all occurrences of CREATED_TIME_ID in the Core Framework source code. (OE0529)
   1. **N/A**: CREATED_TIME_ID does not occur in the source code.

B. Determine if CREATED_TIME_ID is implemented as a property. (OE0529)
   1. **Pass:** A property with an id of CREATED_TIME_ID is implemented.
   2. **N/A:** A property with an id of CREATED_TIME_ID is not implemented.

C. Verify that the value field associated with the CREATED_TIME_ID property is an unsigned long long. (OE0529)
   1. **Pass:** The value field is an unsigned long long.
   2. **Fail:** The value field is not an unsigned long long.

D. Verify that the time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0529)
   1. **Pass**: The time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970.
   2. **Fail**: The time provided is not the number of seconds since 00:00:00 UTC, Jan. 1, 1970.

Manual Test Steps

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

OE_TC_109				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all occurrences of CREATED_TIME_ID in the OE source code. (OE0529)				
1. Search the source code for all occurrences of CREATED_TIME_ID and record the files in which it occurs.	**N/A**: CREATED_TIME_ID does not occur in the source code. (OE0529)		Since code generation from the CF.idl will result in at least one occurrence of CREATED_TIME_ID, this should never happen if we have all their source code.	
B. Determine if CREATED_TIME_ID is implemented as a property. (OE0529)				
2. Examine the occurrences of CREATED_TIME_ID and determine if a property with an id of CREATED_TIME_ID is implemented	**Pass**: A property with the id of CREATED_TIME_ID is implemented. (OE0529) **N/A**: A property with the id of CREATED_TIME_ID is not implemented. (OE0529)		ctime is a standard name for this data in an OMG file description.	
C. Verify that the value field associated with the CREATED_TIME_ID property is an unsigned long long. (OE0529)				
3. Locate the definition of the property with an id of CREATED_TIME_ID.	**Pass**: A definition of the property with an id of CREATED_TIME_ID is found. (OE0529) **Fail**: A definition of the property with an id of CREATED_TIME_ID is not found. (OE0529)			

OE_TC_109				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the value field associated with this property is of type unsigned long long.	**Pass**: The value field is of type unsigned long long. (OE0529) **Fail**: The value field is not of type unsigned long long. (OE0529)		The type unsigned long long is a 64-bit integer field. If the definition of value forces a 64-bit integer field, then this step passes. Note that this will probably be within the *list* operation of the *FileSystem*.	
D. Verify that the time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0529)				
5. Determine the operation provides the time and determine if it is an OS call or within the Core Framework. If it is an OS operation, go to step 7.	The operation is found.		This will most likely be an OS operation, but may be done within the *FileSystem*.	
6. Verify that the operation sets time to the number of seconds since 00:00:00 UTC, Jan. 1, 1970.	**Pass**: The time is set to the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0529) **Fail**: The time is not set to the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0529)			
7. Verify in the OS documentation for its operation that the time returned is the number of seconds since 00:00:00 UTC, Jan. 1, 1970.	**Pass**: The time returned is the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0529) **Fail**: The time returned is not the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0529)			
End of Test				

Test Recording Log – OE_TC_109					
Step 1 (files)	Step 2 (property implemented)	Step 3 (property definition)	Step 4 (type unsigned long long)	Step 5 (Operation that provides time	Step 6/7 (time is provided in UTC format)

Test Summary OE_TC_109

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

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

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

OE0529_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.15. OE_TC_110 - FileSystem's MODIFIED_TIME_ID property

Test Case Number: OE_TC_110

FileSystem

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0530	For this property, the identifier is MODIFIED_TIME_ID and the value shall be an unsigned long long data type containing the number of seconds since 00:00:00 UTC, Jan. 1, 1970.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-79, Figure 3-30; Page 3-81, Section 3.1.3.4.2.3.7

Test Objective

This test case verifies OE0530. The objective of this test is to verify that the MODIFIED_TIME_ID property is implemented and that the data type is correct.

Places to Verify

FileSystem

IDL References

Data

```
const string MODIFIED_TIME_ID = " MODIFIED_TIME";

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

Preconditions

- The Core Framework source code files are available.

Test Description

A. Locate all occurrences of MODIFIED_TIME_ID in the Core Framework source code. (OE0530)
   1. **N/A**: MODIFIED_TIME_ID does not occur in the source code.

B. Determine if MODIFIED_TIME_ID is implemented as a property. (OE0530)
   1. **Pass:** A property with an id of MODIFIED_TIME_ID is implemented.
   2. **N/A:** A property with an id of MODIFIED_TIME_ID is not implemented.

C. Verify that the value field associated with the MODIFIED_TIME_ID property is an unsigned long long. (OE0530)
   1. **Pass:** The value field is an unsigned long long.
   2. **Fail:** The value field is not an unsigned long long.

D. Verify that the time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0530)
   1. **Pass**: The time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970.
   2. **Fail**: The time provided is not the number of seconds since 00:00:00 UTC, Jan. 1, 1970.

Manual Test Steps

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

OE_TC_110				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all occurrences of MODIFIED_TIME_ID in the OE source code. (OE0530)				
1. Search the source code for all occurrences of MODIFIED_TIME_ID and record the files in which it occurs.	**N/A**: MODIFIED_TIME_ID does not occur in the source code. (OE0530)		Since code generation from the CF.idl will result in at least one occurrence of MODIFIED_TIME_ID, an N/A result should never happen if we have all their source code.	
B. Determine if MODIFIED_TIME_ID is implemented as a property. (OE0530)				
2. Examine the occurrences of MODIFIED_TIME_ID and determine if a property with an id of MODIFIED_TIME_ID is implemented	**Pass**: A property with the id of MODIFIED_TIME_ID is implemented. (OE0530) **N/A**: A property with the id of MODIFIED_TIME_ID is not implemented. (OE0530)		ctime is a standard name for this data in an OMG file description.	
C. Verify that the value field associated with the MODIFIED_TIME_ID property is an unsigned long long. (OE0530)				
3. Locate the definition of the property with an id of MODIFIED _TIME_ID.	**Pass**: A definition of the property with an id of MODIFIED_TIME_ID is found. (OE0530) **Fail**: A definition of the property with an id of MODIFIED_TIME_ID is not found. (OE0530)			

OE_TC_110				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the value field associated with this property is of type unsigned long long.	**Pass**: The value field is of type unsigned long long. (OE0530) **Fail**: The value field is not of type unsigned long long. (OE0530)		The type unsigned long long is a 64-bit integer field. If the definition of value forces a 64-bit integer field, then this step passes. Note that this will probably be within the *list* operation of the *FileSystem*.	
D. Verify that the time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970 (OE0530)				
5. Determine the operation provides the time and determine if it is an OS call or within the Core Framework. If it is an OS operation go to step 7.	The operation is found.		This will most likely be an OS operation, but may be done within the *FileSystem*.	
6. Verify that the operation sets time to the number of seconds since 00:00:00 UTC, Jan. 1, 1970.	**Pass**: The time is set to the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0530) **Fail**: The time is not set to the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0530)			
7. Verify in the OS documentation for its operation that the time returned is the number of seconds since 00:00:00 UTC, Jan. 1, 1970.	**Pass**: The time returned is the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0530) **Fail**: The time returned is not the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0530)			
End of Test				

Test Recording Log – OE_TC_110					
Step 1 (files)	Step 2 (property implemented)	Step 3 (property definition)	Step 4 (type unsigned long long)	Step 5 (Operation that provides time	Step 6/7 (time is provided in UTC format)

Test Summary OE_TC_110

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

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

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

OE0530_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.16. OE_TC_111 - FileSystem's LAST_ACCESS_TIME_ID property

Test Case Number: OE_TC_111

FileSystem

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0531	For this property, the identifier is LAST_ACCESS_TIME_ID and the value shall be an unsigned long long data type containing the number of seconds since 00:00:00 UTC, Jan. 1, 1970.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-79, Figure 3-30; Page 3-81, Section 3.1.3.4.2.3.8

Test Objective

This test case verifies OE0531. The objective of this test is to verify that the LAST_ACCESS_TIME_ID property is implemented and that the data type is correct.

Places to Verify

FileSystem

IDL References

Data

```
const string LAST_ACCESS_TIME_ID = "LAST_ACCESS_TIME";

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

Preconditions

- The Core Framework source code files are available.

Test Description

A. Locate all occurrences of LAST_ACCESS_TIME_ID in the Core Framework source code. (OE0531)
   1. **N/A**: LAST_ACCESS_TIME_ID does not occur in the source code.

B. Determine if LAST_ACCESS_TIME_ID is implemented as a property. (OE0531)
   1. **Pass:** A property with an id of LAST_ACCESS_TIME_ID is implemented.
   2. **N/A:** A property with an id of LAST_ACCESS_TIME_ID is not implemented.

C. Verify that the value field associated with the LAST_ACCESS_TIME_ID property is an unsigned long long. (OE0531)
   1. **Pass:** The value field is an unsigned long long.
   2. **Fail:** The value field is not an unsigned long long.

D. Verify that the time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0531)
   1. **Pass**: The time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970.
   2. **Fail**: The time provided is not the number of seconds since 00:00:00 UTC, Jan. 1, 1970.

Manual Test Steps

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

OE_TC_111				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all occurrences of LAST_ACCESS_TIME_ID in the OE source code. (OE0531)				
1. Search the source code for all occurrences of LAST_ACCESS_TIME_ID and record the files in which it occurs.	**N/A**: LAST_ACCESS_TIME_ID does not occur in the source code. (OE0531)		Since code generation from the CF.idl will result in at least one occurrence of LAST_ACCESS_TIME_ID, an N/A result should never happen if we have all their source code.	
B. Determine if LAST_ACCESS_TIME_ID is implemented as a property. (OE0531)				
2. Examine the occurrences of LAST_ACCESS_TIME_ID and determine if a property with an id of LAST_ACCESS_TIME_ID is implemented	**Pass**: A property with the id of LAST_ACCESS_TIME_ID is implemented. (OE0531) **N/A**: A property with the id of LAST_ACCESS_TIME_ID is not implemented. (OE0531)		ctime is a standard name for this data in an OMG file description.	
C. Verify that the value field associated with the LAST_ACCESS_TIME_ID property is an unsigned long long. (OE0531)				
3. Locate the definition of the property with an id of LAST_ACCESS_TIME_ID.	**Pass**: A definition of the property with an id of LAST_ACCESS_TIME_ID is found. (OE0531) **Fail**: A definition of the property with an id of LAST_ACCESS_TIME_ID is not found. (OE0531)			

OE_TC_111				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the value field associated with this property is of type unsigned long long.	**Pass**: The value field is of type unsigned long long. (OE0531) **Fail**: The value field is not of type unsigned long long. (OE0531)		The type unsigned long long is a 64-bit integer field. If the definition of value forces a 64-bit integer field, then this step passes. Note that this will probably be within the *list* operation of the *FileSystem*.	
D. Verify that the time provided is the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0531)				
5. Determine the operation provides the time and determine if it is an OS call or within the Core Framework. If it is an OS operation go to step 7.	The operation is found.		This will most likely be an OS operation, but may be done within the *FileSystem*.	
6. Verify that the operation sets time to the number of seconds since 00:00:00 UTC, Jan. 1, 1970	**Pass**: The time is set to the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0531) **Fail**: The time is not set to the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0531)			
7. Verify in the OS documentation for its operation that the time returned is the number of seconds since 00:00:00 UTC, Jan. 1, 1970.	**Pass**: The time returned is the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0531) **Fail**: The time returned is not the number of seconds since 00:00:00 UTC, Jan. 1, 1970. (OE0531)			
End of Test				

Test Recording Log – OE_TC_111					
Step 1 (files)	**Step 2** (property implemented)	**Step 3** (property definition)	**Step 4** (type unsigned long long)	**Step 5** (Operation that provides time	**Step 6/7** (time is provided in UTC format)

Test Summary OE_TC_111

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

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

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

OE0531_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.17. OE_TC_114 - FileSystem :: mkdir

Test Case Number: OE_TC_114

FileSystem::mkdir

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0560	The *mkdir* operation shall create a file system directory based on the *directoryName* given.
OE0561	The *mkdir* operation shall create all parent directories required to create the directoryName path given.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-84, Section 3.1.3.4.2.5.7.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-8

Test Objective

This test case verifies requirement OE0560 and OE0561. The objective of this test is to verify that the *mkdir* operation creates a file system directory based on the *directoryName* parameter. In addition, when necessary the mkdir operation creates all required parent directories to accomplish that.

Places to Verify

FileSystem and FileManager

IDL References

Exceptions

```
exception InvalidFileName {CF::ErrorNumberType errorNumber; string msg;};
exception FileException {CF::ErrorNumberType errorNumber; string msg;};
```

Operations

```
void mkdir (in string directoryName) raises (CF::InvalidFileName, CF::FileException);
```

Preconditions

- OE source code files are available for all FileSystem and FileManager files having the *mkdir* operation.

Test Description

For each FileSystem and FileManager:

A. Verify that the *mkdir* operation is found in the source code and the processing of the *directoryName* parameter is found within it. (OE0560, OE0561)
   1. **Pass**: The *mkdir* operation is found in the source code and the processing of the input parameter is found within it.
   2. **Untested**: The *mkdir* is not found in any source code.
   3. **Fail**: The processing of the input parameter is not found in any source code.

B. Verify that the *mkdir* operation uses the parameter *directoryName* to create all required parent directories. (OE0561)
   1. **Pass**: The *mkdir* operation uses the parameter *directoryName* to create all required parent directories.
   2. **Fail**: The *mkdir* operation does not use the parameter *directoryName* to create all required parent directories.

C. Verify that the *mkdir* operation uses the parameter *directoryName* to create a file system directory. (OE0560)
   1. **Pass**: The *mkdir* operation uses the parameter *directoryName* to create a file system directory.
   2. **Fail**: The *mkdir* operation does not use the parameter *directoryName* to create a file system directory.

Manual Test Steps

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

OE_TC_114				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
For each FileSystem and FileManager:				
A. Verify that the source code for the *mkdir* operation is found (OE0560, OE0561)				
1. Locate the source code file having the *mkdir* operation.	**Pass**: The *mkdir* operation is found in the source code.(OE0560, OE0561) **Untested**: The *mkdir* operation is not found in any source code. End of Test. (OE0560, OE0561)		Multiple FileSystems may exist. int mkdirResult = ::mkdir ((char*) DIRECTORY_NAME, mode);	
2. Find in the source code of the *mkdir* operation(s) where the *directoryName* parameter is processed.	**Pass**: The processing of the *directoryName* parameter is found. (OE0560, OE0561) **Fail**: The processing of the *directoryName* parameter is not found. (OE0560, OE0561)		For most FileSystems a *directoryName* parameter needs to be processed into pathnames separately. Possible methods would be to parse the pathname or possibly use a recursive routine to determine the proper path.	
B. Verify that the FileManager *mkdir* operation uses the parameter *directoryName* to create all parent directories. (OE0561)				

OE_TC_114				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify the FileManager *mkdir* operation uses the processed directory names taken from the *directoryName* parameter to create directories that do not exist.	**Pass**: The FileManager *mkdir* operation uses the processed directory names taken from the *directoryName* parameter to create directories that do not exist. (OE0561) **Fail**: The FileManager *mkdir* operation does not use the processed directory names taken from the *directoryName* parameter to create directories that do not exist. (OE0561)		void FileManager_Servant::*mkdir*(const char* directoryName CF_ENV_ARG_DECL)	
C. Verify that the FileSystem *mkdir* operation uses the parameter *directoryName* to create a file system directory. (OE0560)				
4. Verify the FileSystem *mkdir* operation uses all the processed names from the *directoryName* parameter to create a new directory.	**Pass**: The FileSystem *mkdir* operation uses all the processed names from the parameter *directoryName* to create a file system directory. (OE0560) **Fail**: The FileSystem *mkdir* operation does not use all the processed names from the parameter *directoryName* to create a file system directory. (OE0560)		CORBA::String_var pathInFileSystem; CF::FileSystem_var fileSystem = resolveFileSystemReference(directoryName, pathInFileSystem.out()); if (!CORBA::is_nil(fileSystem)) { fileSystem->mkdir(pathInFileSystem.in() CF_ENV_ARG_PARAMETER);	
End of Test				

Test Recording Log – OE_TC_114		
Step 1 & 2 (source code files having *mkdir*)	**Step 3** (FileManager ***mkdir*** uses ***directoryName*** *to create parent directories*)	**Step 4** (FileSystem ***mkdir*** uses ***directoryName*** *to create file system directory*)

Test Summary OE_TC_114

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

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

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

OE0560 _____

OE0561 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.18. OE_TC_115 - FileSystem :: rmdir

Test Case Number: OE_TC_115

FileSystem::rmdir

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0564	The *rmdir* operation shall remove the directory identified by the input *directoryName* parameter.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-85, Section 3.1.3.4.2.5.8.3

Test Objective

This test case verifies requirement OE0564. The objective of this test is to verify that the *rmdir* operation removes the directory identified by the input parameter *directoryName*. Assumption is that the directory identified by the input *directoryName* parameter is empty. This test case does not check if the directory has any files and/or directories.

Places to Verify

FileSystem

IDL References

Exceptions

CF FileException, CF InvalidFileName

Operations

void *rmdir* (in string directoryName) raises (InvalidFileName, FileException);

Preconditions

- Source code files with the *rmdir* operation for the OE under test are available.
- OE directories exist and are available.

Test Description

For each FileSystem, perform the following:

A. Verify that the *rmdir* operation is found in the source code. (OE0564)
   1. **Pass**: An implementation of the *rmdir* operation is found in the source code.
   2. **Untested**: An implementation of *rmdir* operation is not found.

B. Verify that the *rmdir* operation checks that the input parameter *directoryName* is a directory in the FileSystem. (OE0564)
   1. **Pass**: The directory identified by the input parameter *directoryName* exists in the FileSystem**.**
   2. **Untested**: The directory identified by the input parameter *directoryName* does not exist in the FileSystem.
   3. **N/A:** The directory identified by the input parameter *directoryName* exists in the FileSystem and contains files**.**

C. Verify that the *rmdir* operation removes the directory identified by the input parameter *directoryName*. (OE0564)
   1. **Pass**: The directory identified by the input parameter *directoryName* is removed from the FileSystem**.**
   2. **Fail:** The directory identified by the input parameter *directoryName* is not removed from the FileSystem.

Manual Test Steps

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

OE_TC_115				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Verify that the *rmdir* operation is found in the source code. (OE0564).				
1. Locate the source code file that implements the *rmdir* operation of the FileSystem.	**Pass**: An implementation of the *rmdir* operation is found in the source code. (OE0564) **Untested**: An implementation of *rmdir* operation is not found. (OE0564)		** There may be multiple FileSystems.	
B. Verify that the *rmdir* operation checks that the input parameter *directoryName* is a directory in the FileSystem.				
2. Verify that the *rmdir* operation checks that the input parameter *directoryName* is a directory in the FileSystem.	**Pass**: The directory identified by the input parameter *directoryName* exists in the FileSystem. (OE0564) **Untested**: The directory identified by the input parameter *directoryName* does not exist in the FileSystem. (OE0564) **N/A:** The directory identified by the input parameter *directoryName* exists in the FileSystem and contains files. (OE0564)		The *directoryName* parameter is used to determine the mount point name and pathname.	
C. Verify that the *rmdir* operation removes the directory identified by the input parameter *directoryName*. (OE0564)				

OE_TC_115				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *rmdir* operation removes the directory identified by the input parameter *directoryName*.	**Pass**: The directory identified by the input parameter *directoryName* is removed from the FileSystem. (OE0564) **Fail:** The directory identified by the input parameter *directoryName* is not removed from the FileSystem. (OE0564)			
End of Test				

Test Recording Log – OE_TC_115		
Step 1 (source code implementing *rmdir*)	**Step 2** (*directoryName* is a directory)	**Step 3** (*rmdir* operation removes *directoryName)*

Test Summary OE_TC_115

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

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

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

OE0564 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.4.19. OE_TC_136 - FileSystem :: remove raises InvalidFileName

Test Case Number: OE_TC_136

FileSystem::*remove* raises InvalidFileName
FileManager::*remove* raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0533	The *remove* operation shall raise the CF InvalidFileName exception when the input fileName parameter is not a valid absolute pathname.
OE0599	The CF InvalidFileName exception indicates an invalid file name was passed to a file service operation. The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-81, Section 3.1.3.4.2.5.1.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 – C-4,C-7

Test Objective

This test case verifies OE0533 and OE0599. The objective of this test is to verify that the *remove* operation raises the *CF::InvalidFileName* exception when the input filename parameter is invalid or not an absolute path name. This test case also verifies that the exception must include an error number of *CF::ErrorNumberType*.

Places to Verify

File systems and the file manager of the Core Framework.

IDL References

Data

enum ErrorNumberType { CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP ,CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE , CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT ,CF_EXDEV };

Exceptions

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

Operations

void *remove* (in string fileName) raises (CF::FileException, CF::InvalidFileName);

Preconditions

- The source code files for the FileSystem and FileManager of the Core Framework are available.

Test Description

A. Identify the implementations of the *remove* operation in the Core Framework. (OE0533)
   1. **Fail**: There are no implementations of the *remove* operation in the Core Framework.

For every implementation of the *remove* operation in the Core Framework, perform the following steps:

B. Verify that the *remove* operation raises the *CF::InvalidFileName* exception when the input fileName parameter is not a valid absolute pathname.(OE0533)
   1. **Pass**: The *remove* operation raises the *CF::InvalidFileName* exception when the input filename parameter is not a valid absolute pathname.
   2. **Fail**: The *remove* operation does not raise the *CF::InvalidFileName* exception when the input filename parameter is not a valid absolute pathname.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)
   1. **Pass:** The *CF::InvalidFileName* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The *CF::InvalidFileName* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_136				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the *remove* operation in the Core Framework. (OE0533)				
1. Examine the source code files of the Core Framework and identify the implementations of the *remove* operation in the Core Framework. List the source code files that implement the *remove* operation.	**Fail**: There are no implementations of the *remove* operation in the Core Framework. (OE0533)		The *remove* operation will most likely be located in the file systems and the file manager of the Core Framework. A Core Framework may contain an empty "virtual" function for the *remove* operation and an actual implementation that the operation uses. It is only the *remove* operations that are actually implemented that are to be tested. The operation's functionality may be implemented through inheritance, this is acceptable.	
For every implementation of the *remove* operation in the Core Framework, perform the following steps:				
B. Verify that the *remove* operation raises the *CF::InvalidFileName* exception when the input fileName parameter is not a valid absolute pathname. (OE0533)				

OE_TC_136				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
2. Verify that the *CF::InvalidFileName* exception is raised by the *remove* operation when the fileName parameter does not start with a forward slash, representing an absolute path.	**Pass**: The *remove* operation raises the *CF::InvalidFileName* exception when the input filename parameter is not a valid absolute pathname. (OE0533) **Fail**: The *remove* operation does not raise the *CF::InvalidFileName* exception when the input filename parameter is not a valid absolute pathname. (OE0533)		Valid pathnames are structured according to the POSIX specification whose valid characters include the "/" (forward slash) character in addition to the valid filename characters. (A valid pathname may consist of a single filename.) An "absolute pathname" is a pathname which starts with a(forward slash) "/" character.(Page 1-2)	
3. Verify that the *CF::InvalidFileName* exception is raised by the *remove* operation when the fileName parameter does not contain a valid directory pathname that exists in the file system.	**Pass**: The *remove* operation raises the *CF::InvalidFileName* exception when the input filename parameter does not contain a valid directory pathname that exists in the file system. (OE0533) **Fail**: The *remove* operation does not raise the *CF::InvalidFileName* exception when the input filename parameter does not contain a valid directory pathname that exists in the file system. (OE0533)			

OE_TC_136				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the *CF::InvalidFileName* exception is raised by the *remove* operation when the fileName parameter contains a filename at the end of the path that does not exist in the file system.	**Pass**: The *remove* operation raises the *CF::InvalidFileName* exception when the input filename parameter contains a filename at the end of the path that does not exist in the file system. (OE0533) **Fail**: The *remove* operation does not raise the *CF::InvalidFileName* exception when the input filename parameter contains a filename at the end of the path that does not exist in the file system. (OE0533)			
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)				
5. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_136					
Step 1 (source file name)	**Step 2** (absolute path/forward slash?)	**Step 3** (directory exists?)	**Step 4** (filename exists?)	**Step 5** (ErrorNumberType)	**Notes**

Test Summary OE_TC_136

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

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

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

OE0533 _____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.20. OE_TC_137 - FileSystem :: copy raises InvalidFileName when inputs are invalid

Test Case Number: OE_TC_137

FileSystem::copy raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0537	The *copy* operation shall raise the CF InvalidFileName exception when the sourceFileName or destinationFileName input parameters are not a valid absolute pathnames.
OE0599	The CF InvalidFileName exception indicates an invalid file name was passed to a file service operation. The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-82, Section 3.1.3.4.2.5.2.5 Page 1-2, Section 1.3.1.1; Page 3-79, Section 3.1.3.4.2.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 and C-4, C-7

Test Objective

This test case verifies OE0537 and OE0599. The objective of this test is to (1) verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *sourceFileName* or *destinationFileName* input parameters are not a valid absolute pathname (2) verify that the error number of the exception is of the type, CF::ErrorNumberType.

Places to Verify

File systems and the file manager of the Core Framework

IDL References

Data

enum ErrorNumberType {
CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR, CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT,

CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV};

Exceptions

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

Operations

void copy (in string sourceFileName, in string destinationFileName) raises (CF::InvalidFileName, CF::FileException);

Preconditions

- The source code files of the file systems and file managers of the Core Framework are available.

Test Description

A. Identify the implementations of the *copy* operation for FileSystem and FileManager in the Core Framework. (OE0537)
   1. **Pass**: There is some implementation of the *copy* operation for FileSystem or FileManager in the Core Framework.
   2. **Fail**: There are no implementations of the *copy* operation for FileSystem or FileManager in the Core Framework.

For each implementation of the *copy* operation in the Core Framework, perform the following steps:

B. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when either the *sourceFileName* or *destinationFileName* input parameters are not a valid absolute pathname. (OE0537)
   1. **Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when either the *sourceFileName* or *destinationFileName* input parameters are not a valid absolute pathname.
   2. **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when either the *sourceFileName* or *destinationFileName* input parameters are not a valid absolute pathname.

C. Verify that the error number, which accompanies the exception, is a CF::ErrorNumberType value. (OE0599)
   1. **Pass:** The *CF::InvalidFileName* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The CF::InvalidFileName exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_137				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the copy operation for FileSystem and FileManager in the Core Framework. (OE0537)				
1. Examine the source code files of the FileSystems and the FileManager of the Core Framework and identify the implementations of the *copy* operation. List the source code where the *copy* operation is implemented.	**Pass**: There are some implementations of the *copy* operation in the Core Framework. (OE0537) **Fail**: There are no implementations of the *copy* operation in the Core Framework. (OE0537)		The places where the copy operation is called do not need to be determined, only where the operation is actually declared and implemented. A key word search using a semi-automated tool such as an IDE would be the easiest way to find the implementations of the copy operation in the Core Framework. There may be virtual functions in the source code that are empty, it is only the implementations that need to be examined.	
For each implementation of the *copy* operation in the Core Framework, perform the following steps:				
B. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when either the *sourceFileName* or *destinationFileName* input parameters are not a valid absolute pathname. (OE0537)				

OE_TC_137				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
2. Examine the source code of the *copy* operation and identify the location(s) where the *sourceFileName* parameter is verified.	**Pass:** The location where the *sourceFileName* parameter is verified is located. (OE0537) **Fail:** The location where the *sourceFileName* parameter is verified is not located. (OE0537)			
3. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *sourceFileName* parameter is not an absolute pathname.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the *sourceFileName* parameter is not an absolute pathname. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the *sourceFileName* parameter is not an absolute pathname. (OE0537)		Section 1.3.1.1 page 1-2 "An "absolute pathname" is a pathname which starts with a (forward slash) character"	
4. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *sourceFileName* parameter contains invalid characters as described in the following definition from the SCA v2.2.2. Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *sourceFileName* parameter contains invalid characters. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *sourceFileName* parameter contains invalid characters. (OE0537)		Section 3.1.3.4.2.1 Page 3-79 "Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters. The filenames "." ("dot") and ".." ("dot-dot") are invalid in the context of a file system."	

OE_TC_137				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *sourceFileName* parameter contains either a "." (dot) or a "..", which are invalid in the context of a file system.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *sourceFileName* parameter contains either a single dot or two dots as the only characters in the file name (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *sourceFileName* parameter contains a single dot or double dots as the only elements in the filename. (OE0537)			
6. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *sourceFileName* parameter does not conform to the following syntax. A valid pathname shall not exceed 1024 characters.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *sourceFileName* parameter exceeds 1024 characters. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *sourceFileName* parameter exceeds 1024 characters. (OE0537)			
7. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *sourceFileName* parameter contains an individual filename or directory name that does not conform to the following syntax. Valid individual filenames and directory names shall be 40 characters or less.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *sourceFileName* parameter contains an individual filename or directory name that exceeds 40 characters. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *sourceFileName* parameter contains an individual filename or directory name that exceeds 40 characters. (OE0537)			

OE_TC_137				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
8. Examine the source code of the *copy* operation and identify the location(s) where the *destinationFileName* parameter is verified.	**Pass:** The location where the *destinationFileName* parameter is verified is located. (OE0537) **Fail:** The location where the *destinationFileName* parameter is verified is not located. (OE0537)			
9. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *destinationFileName* parameter is not an absolute pathName.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *destinationFileName* parameter is not an absolute pathname. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *destinationFileName* parameter is not an absolute pathname. (OE0537)			
10. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *destinationFileName* parameter contains invalid characters as described in the following definition from the SCA. Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *destinationFileName* parameter contains invalid characters. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *destinationFileName* parameter contains invalid characters. (OE0537)			

OE_TC_137				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
11. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *destinationFileName* parameter contains either a "."(dot) or a "..", which are invalid in the context of a file system.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *destinationFileName* parameter contains either a "."(single dot) or ".."(two dots) as the only characters in the filename. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *destinationFileName* parameter contains a "."(single dot) or ".." double dots as the only elements in the filename. (OE0537)			
12. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *destinationFileName* parameter does not conform to the following syntax. A valid pathname shall not exceed 1024 characters.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *destinationFileName* parameter exceeds 1024 characters. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *sourceFileName* parameter exceeds 1024 characters. (OE0537)			
13. Verify that the *copy* operation raises the *CF::InvalidFileName* exception when the *destinationFileName* parameter contains an individual filename or directorynames that does not conform to the following syntax. Valid individual filenames and directory names shall be 40 characters or less.	**Pass**: The *copy* operation raises the *CF::InvalidFileName* exception when the input *destinationFileName* parameter contains an individual filename or directory name that exceeds 40 characters. (OE0537) **Fail**: The *copy* operation does not raise the *CF::InvalidFileName* exception when the input *destinationFileName* parameter contains an individual filename or directory name that exceeds 40 characters. (OE0537)			
C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0599)				

OE_TC_137				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
14. Determine if the exception includes an error number (errorNumber) of the type: ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. **(**OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_137											
Step1 (file name(s) copy operation)											
sourceFileName					destinationFilename						
Step3 (slash)	Step4 (valid)	Step5 (dot)	Step6 (1024)	Step7 (40)	Step9 (slash)	Step10 (valid)	Step11 (dot)	Step12 (1024)	Step13 (40)	Step14 (Error Number Type?)	Notes

Test Summary OE_TC_137

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

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

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

OE0537 _____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.21. OE_TC_138 - FileSystem :: exists

Test Case Number: OE_TC_138

FileSystem::exists

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0538	The *exists* operation shall check to see if a file exists based on the fileName parameter.
OE0539	The *exists* operation shall return TRUE if the file exists, or FALSE if it does not.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-82, Section 3.1.3.4.2.5.3.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-7, Section C.1

Test Objective

This test case verifies the requirements, OE0538 and OE0539. The objective of this test is to verify that the *exists* operation checks that the file represented by the filename parameter exists in the directory structure and returns the correct value.

Places to Verify

FileSystem, FileManager

IDL References

Operations

boolean exists (in string fileName) raises (CF::InvalidFileName);

Preconditions

- The FileManager and FileSystem source code of the Core Framework is available.

Test Description

A. Identify the implementations of the *exists* operation in the Core Framework (OE0538) (OE0539).
   1. **Fail:** There are no implementations of the *exists* operation in the Core Framework.

B. Verify that the *exists* operation checks that the file represented by the fileName parameter exists in the file system (OE0538).
   1. **Pass**: The *exists* operation checks that the file represented by the fileName parameter exists in file system.
   2. **Fail**: The *exists* operation does not check that the file represented by the filename parameter exists in the file system.

C. **Verify the** *exists* operation returns TRUE if the file exists, or FALSE if it does not (OE0539).
   1. **Pass:** The exists operation returns TRUE if the file exists and FALSE if the file does not exist.
   2. **Fail:** The exists operation does not return TRUE if the file exists or FALSE if the file does not exist.

Manual Test Steps

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

OE_TC_138				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the *exists* operation in the Core Framework (OE0538).				
1. Examine the source code of the Core Framework and identify the implementations of the *exists* operation. Lists the names of the source files having the *exists* operation.	**Untested**: There are no implementations of the *exists* operation in the Core Framework. (OE0538)			
B. Verify that the *exists* operation checks that the file represented by the fileName parameter exists in the file system (OE0538).				
2. Verify that *exists* operation checks that the file represented by the operation's filename parameter exists within the directory structure.	**Pass**: The *exists* operation checks that the file represented by the fileName parameter exists in file system. (OE0538) **Fail**: The *exists* operation does not check that the file represented by the filename parameter exists in the file system. (OE0538)			
C. Verify the exists operation returns TRUE if the file exists, or FALSE if it does not (OE0539)				
3. Verify that TRUE or FALSE is returned based on the results of the operation verified in step 2.	**Pass:** The exists operation returns TRUE if the file exists and FALSE if the file does not exist. (OE0539) **Fail:** The exists operation does not return TRUE if the file exists or FALSE if the file does not exist. (OE0539)			
End of Test				

Test Recording Log – OE_TC_138		
Step 1 (source files containing *exists* operation)	**Step 2** (*exists* operation - file based on the filename parameter exists - (Y/N?)	**Step 3** (result returned)

Test Summary OE_TC_138

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

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

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

0E0538_____

0E0539_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.22. OE_TC_139 - FileSystem :: create raises InvalidFileName

Test Case Number: OE_TC_139

FileSystem::create

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0551	The *create* operation shall raise the CF InvalidFileName exception when the input fileName parameter is not a valid absolute pathname.
OE0599	The CF InvalidFileName exception indicates an invalid file name was passed to a file service operation. The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-83, Section 3.1.3.4.2.5.5.5; Page 1-2, Section 1.3.1.1; Page 3-79, Section 3.1.3.4.2.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 and C-4, C-7

Test Objective

This test case verifies OE0551 and OE0599. The objective of this test is to verify that the *create* operation raises the *CF::InvalidFileName* exception when the input fileName parameter is not a valid absolute pathname(OE0551) and to verify that the error number of the exception is of the type, CF::ErrorNumberType (OE0599). The definition of absolute pathname is taken from the SCA v2.2.2 document and specifies that a pathname must begin with a forward slash. The definition of a valid character is taken from the SCA v2.2.2 document, which means that the pathname may only contain the following characters: the 62 alphanumeric characters, the period, the underscore and the hyphen, although a filename cannot be made up entirely of a single or double dot.

Places to Verify

File systems and file managers of the Core Framework

IDL References

Data

enum ErrorNumberType {

CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR, CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV};

Exceptions

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

Operations

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

Preconditions

- The source code files of the file systems and file managers of the Core Framework are available.

Test Description

A. Identify the implementations of the *create* operation in the Core Framework. (OE0551)
   1. **Fail**: There are no implementations of the *create* operation in the Core Framework.

For each implementation of the *create* operation in the Core Framework, perform the following steps:

B. Verify that the *create* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname. (OE0551)
   1. **Pass**: The *create* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname.
   2. **Fail**: The *create* operation does not raise the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)
   1. **Pass:** The *CF::InvalidFileName* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The *CF::InvalidFileName* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_139				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the *create* operation in the Core Framework. (OE0551)				
1. Examine the source code files of the file systems and file managers of the Core Framework and identify the implementations of the *create* operation. List the source code where the *create* operation is implemented.	**Fail**: There are no implementations of the *create* operation in the Core Framework. (OE0551)		The places where the *create* operation is called do not need to be determined, only where the operation is actually declared and implemented. A key word search using a semi-automated tool such as an IDE would be the easiest way to find the implementations of the *create* operation in the Core Framework. There may be virtual functions in the source code that are empty; it is only the implementations that need to be examined. It is acceptable for the *create* operation to inherit the exception functionality.	
For each implementation of the *create* operation in the Core Framework, perform the following steps:				
B. Verify that the *create* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname. (OE0551)				
2. Examine the source code of the *create* operation and identify the location(s) where the *CF::InvalidFileName* is raised.	The location where the *CF::InvalidFileName* exception is raised is located.			

OE_TC_139				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *create* operation raises the *CF::InvalidFileName* exception when the *fileName* parameter is not an absolute pathname, which is defined by the following statement from the SCA v2.2.2 Section 1.3.1.1 page 1-2 "An "absolute pathname" is a pathname which starts with a / (forward slash) character.	**Pass**: The *create* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname. (OE0551) **Fail**: The *create* operation does not raise the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname. (OE0551)			
4. Verify that the *create* operation raises the *CF::InvalidFileName* exception when the *fileName* parameter contains characters outside of the SCA v2.2.2's definition of being valid, which is defined as having the following constraints. "Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters. The filenames "." ("dot") and ".." ("dot-dot") are invalid in the context of a file system."	**Pass**: The *create* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter contains characters that are not defined by the SCA v2.2.2 as being valid. (OE0551) **Fail**: The *create* operation does not raise the *CF::InvalidFileName* exception when the input *fileName* parameter contains characters that are not defined by the SCA v2.2.2 as being valid. (OE0551)		The definition of the SCA v2.2.2's valid character that is listed in this step's column one originates in Section 3.1.3.4.2.1, Page 3-79 of the SCA document.	
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)				

OE_TC_139				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Determine if the exception includes an error number (errorNumber) of the type: ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. **(**OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. **(**OE0599)			
End of Test				

Test Recording Log – OE_TC_139					
Step 1 (file name(s) of *create* operation) (Pass/Fail?)	**Step 2** (Locations where the exception is raised inside the *create* operation)	**Step 3** (pathname is absolute?) (Pass/Fail?)	**Step 4** (pathname contains only valid chars) (Pass/Fail?)	**Step 5** (CF::ErrorNumberType) (Pass/Fail?)	**Notes**

Test Summary OE_TC_139

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

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

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

OE0551 _____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.4.23. OE_TC_140 - FileSystem :: open raises InvalidFileName

Test Case Number: OE_TC_140

FileSystem::open raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0559	The *open* operation shall raise the CF InvalidFileName exception when the input fileName parameter is not a valid absolute pathname.
OE0599	The CF InvalidFileName exception indicates an invalid file name was passed to a file service operation. The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-84, Section 3.1.3.4.2.5.6.5; Page 1-2, Section 1.3.1.1; Page 3-79, Section 3.1.3.4.2.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 and C-4, C-8

Test Objective

This test case verifies OE0559 and OE0599. The objective of this test is to (1) verify that the *open* operation raises the *CF::InvalidFileName* exception when the input fileName parameter is not a valid absolute pathname, (2) verify that the error number of the exception is of the type, CF::ErrorNumberType.

Note: Related Test Cases are OE_TC_060 (OE0002, OE0605, OE0605-C124, OE0733, OE0733-C125) and OE_TC_136 (OE0533).

Places to Verify

File systems and file managers of the Core Framework

IDL References

Data

```
enum ErrorNumberType {
        CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
        CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
        CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,
```

CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV};

Exceptions

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

Operations

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

Preconditions

- The source code files of the file systems and the file manager having the open operation of the Core Framework are available.

Test Description

A. Identify the implementations of the *open* operation in the Core Framework. (OE0559)
   1. **Fail**: There are no implementations of the *open* operation in the Core Framework.

For each implementation of the *open* operation in the Core Framework, perform the following steps:

B. Verify that the *open* operation raises the *CF::InvalidFileName* exception when the input fileName parameter is not a valid absolute pathname. (OE0559)
   1. **Pass**: The *open* operation raises the *CF::InvalidFileName* exception when the input filename parameter is not a valid absolute pathname.
   2. **Fail**: The *open* operation does not raise the *CF::InvalidFileName* exception when the input filename parameter is not a valid absolute pathname.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)
   1. **Pass:** The *CF::InvalidFileName* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The *CF::InvalidFileName* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_140				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the *open* operation in the Core Framework. (OE0559)				
1. Examine the source code files of the file systems and file managers having the *open* operation. List the source code file name(s).	**Fail**: There are no implementations of the *open* operation in the Core Framework. (OE0559)		The places where the *open* operation is called do not need to be determined, only where the operation is actually declared and implemented. A key word search using a semi-automated tool such as an IDE would be the easiest way to find the implementations of the *open* operation in the Core Framework. There may be virtual functions in the source code that are empty, it is only the implementations that need to be examined.	
2. Identify the location(s) where the *CF::InvalidFileName* is raised by the *open* operation.	The location where the *CF::InvalidFileName* exception is raised is located.			
For each implementation of the *open* operation in the Core Framework, perform the following steps:				
B. Verify that the *open* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname. (OE0559)				

OE_TC_140				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *open* operation raises the *CF::InvalidFileName* exception when the fileName parameter is not an absolute pathname.	**Pass**: The *open* operation raises the *CF::InvalidFileName* exception when the input filename parameter is not a valid absolute pathname. (OE0559) **Fail**: The *open* operation does not raise the *CF::InvalidFileName* exception when the input filename parameter is not a valid absolute pathname. (OE0559)		Section 1.3.1.1 page 1-2 "An "absolute pathname" is a pathname which starts with a(forward slash) character"	
4. Verify that the *open* operation raises the *CF::InvalidFileName* exception when the *fileName* parameter contains characters outside of the SCA v2.2.2's definition of being valid, which is defined as having the following constraints. "Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters. The filenames "." ("dot") and ".." ("dot-dot") are invalid in the context of a file system."	**Pass**: *The open* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter contains characters that are not defined by the SCA v2.2.2 as being valid. (OE0559) **Fail**: The *open* operation does not raise the *CF::InvalidFileName* exception when the input *fileName* parameter contains characters that are not defined by the SCA v2.2.2as being valid. (OE0559)		The definition of the SCA v2.2.2's valid character that is listed in this step's column one originates in Section 3.1.3.4.2.1, Page 3-79 of the SCA document.	
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)				

OE_TC_140				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_140			
Step 1 (source file with *open* operation)	**Step 3** (*FileException* raised when absolute pathname is invalid- Y/N?)	**Step 4** (pathname contains only valid chars) (Y/N?)	**Step 5** (*FileException* includes ErrorNumberType- Y/N?)

Test Summary OE_TC_140

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

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

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

OE0559 _____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.4.24. OE_TC_141 - FileSystem :: mkdir raises InvalidFileName

Test Case Number: OE_TC_141

FileSystem::mkdir raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0563	The *mkdir* operation shall raise the CF InvalidFileName exception when the directoryName is not a valid directory name.
OE0599	The CF InvalidFileName exception indicates an invalid file name was passed to a file service operation. The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-84, Section 3.1.3.4.2.5.7.5 Page 3-93, Section 3.1.3.6.4; Page 1-2, Section 1.3.1.1; Page 3-79, Section 3.1.3.4.2.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3, C-4, & C-8

Test Objective

This test case verifies OE0563 and OE0599. The objective of this test case is to verify the *mkdir* operation for the specific situations

(1) raises the *CF::InvalidFileName* exception when the input parameter – *directoryName*, is not a valid directory name (OE0563),

(2) provides an error number with the exception of type, CF::ErrorNumberType (OE0599).

The Methodology of determining a valid pathname, individual file and/or directory name:

A valid pathname is defined by the SCA, and mirrors the POSIX specification, as the only allowable and valid characters to be the "/"(forward slash), the 62 alphanumeric characters, the period, the "_"(underscore) and the "-"(hyphen) with the exception that a filename cannot be made up entirely of a single or double dot.

Also defined in Section 3.1.3.4.2.1, Pg. 3-79 of the SCA, are the length limitations for a valid SCA pathname. A filename must not exceed the length of 40 characters and the pathname must not exceed the length of 1024 characters.

An "**absolute pathname**" is a pathname which starts with a "/" (forward slash) character. Whereas, a "**relative pathname**" does not have a leading "/" character.

Places to Verify

File systems and the file manager of the Core Framework

IDL References

Data

enum ErrorNumberType {
CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR, CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV};

Exceptions

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

Operations

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

Preconditions

- The source code files of the file systems and the file manager of the Core Framework are available.

Test Description

A. Within the Core Framework source code, search for all implementations of the *mkdir* operation. (OE0563). Make a note of each implementation and location that will be used throughout this test case.
   1. **Fail**: There are no implementations of the *mkdir* operation in the Core Framework.

For each implementation (notes from the previous step) of the *mkdir* operation in the Core Framework, perform the remaining steps:

B. Verify the *mkdir* operation raises the *CF::InvalidFileName* exception when the input *directoryName* parameter is not a valid directory name. (OE0563) See the "The Methodology of determining a valid pathname, individual file and/or directory name" above to ascertain the validity of a directory name.
   1. **Pass**: The *mkdir* operation raises the *CF::InvalidFileName* exception when the input *directoryName* parameter is not a valid directory name.
   2. **Fail**: The *mkdir* operation does not raise the *CF::InvalidFileName* exception when the input *directoryName* parameter is not a valid directory name.

C. From the exception received above, verify the error number is a CF::ErrorNumberType value. (OE0599) See the "ErrorNumberType," listed above, that describes acceptable values for CF::ErrorNumberType
   1. **Pass:** The *CF::InvalidFileName* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The *CF::InvalidFileName* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Manual Test Steps

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

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

OE_TC_141				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Within the Core Framework source code, search for all implementations of the *mkdir* operation.. (OE0563)				
1. From the source code files, identify all implementations for the *mkdir* operation within the FileSystem and the FileManager sections of the Core Framework. Make a list of the source code were the *mkdir* operation is implemented.	**Fail**: There are no implementations of the *mkdir* operation in the Core Framework. (OE0563)		The places where the *mkdir* operation is called do not need to be determined, only where the operation is actually declared and implemented. A key word search using a semi-automated tool such as an IDE would be the easiest way to find the implementations of the *mkdir* operation in the Core Framework. There may be virtual functions in the source code that are empty, it is only the implementations that need to be examined. It is acceptable for the *mkdir* operation to inherit the exception functionality.	
For each implementation of the *mkdir* operation in the Core Framework, perform the following steps:				
B. Verify the *mkdir* operation raises the *CF::InvalidFileName* exception, when the input *directoryName* parameter, is not a valid directory name. (OE0563) See the "The Methodology of determining a valid pathname, individual file and/or directory name" above to ascertain the validity of a directory name.				
2. By examining the source code of the mkdir operation, identify the location(s), in source code, where the CF::InvalidFileName is raised.	The location where the *CF::InvalidFileName* exception is raised.		This information is used for the next step.	

OE_TC_141				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Verify that the *mkdir* operation raises the *CF::InvalidFileName* exception when the input *directoryName* parameter is not valid directory name	**Pass**: The *mkdir* operation raises the *CF::InvalidFileName* exception when the input *directoryName* parameter is not a valid directory name because it is not an absolute pathname. (OE0563) **Fail**: The *mkdir* operation does not raise the *CF::InvalidFileName* exception when the input *directoryName* parameter is not a valid directory name because it is not a valid absolute pathname. (OE0563)		Section 1.3.1.1 page 1-2, of the SCA "An "absolute pathname" is a pathname which starts with a "/" (forward slash) character".	
4. Verify that the *mkdir* operation raises the *CF::InvalidFileName* exception when the input *directoryName* contains invalid characters, those outside of the valid characters as defined by the SCA v2.2.2. See the note in the comments column.	**Pass**: The *mkdir* operation raises the *CF::InvalidFileName* exception when the input *directoryName* parameter contains one or more invalid characters. (OE0563) **Fail**: The *mkdir* operation does not raise the *CF::InvalidFileName* exception when the input *directoryName* parameter contains one or more invalid characters (OE0563)		The following quote taken from the SCA v2.2.2 and defines what the valid characters are for an SCA pathname "Valid pathnames are structured according to the POSIX specification whose valid characters include the "/" (forward slash) character in addition to the valid filename characters. Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters. The filenames "." ("dot") and ".." ("dot-dot") are invalid in the context of a file system."(Page 3-79, Section 3.1.3.4.2.1)	
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)				

OE_TC_141				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Determine if the exception includes an error number (errorNumber) of the type: ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_141					
Step 1 (file names)	**Step 2** (*CF::InvalidFileName* location)	**Step 3** (absolute or valid pathname) (Pass/Fail?)	**Step 4** (valid) (Pass/Fail?)	**Step 5** (error number is of CF::ErrorNumberType) (Pass/Fail?)	**Notes**

Test Summary OE_TC_141

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

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

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

OE0563 _____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.25. OE_TC_142 - FileSystem :: rmdir raises InvalidFileName

Test Case Number: OE_TC_142

FileSystem::rmdir raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0566	The *rmdir* operation shall raise the CF InvalidFileName exception when the input directoryName parameter is not a valid path prefix.
OE0599	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 1-2, Section 1.3.1.1 Page 3-85, Section 3.1.3.4.2.5.8.5 Page 3-93, Section 3.1.3.6.4
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	C-3 to C-4, C-8

Test Objective

This test case verifies OE0566 and OE0599. The objective of this test is to verify that the FileSystem *rmdir* operation raises the *InvalidFileName* exception when the input directoryName parameter is not a valid path prefix. Furthermore, verify the exception contains an error number of the type CF::ErrorNumberType. In SCA v2.2.2, section 1.3.1.1, it states 'A "path prefix" is a pathname which refers to a directory and thus does not include the name of a plain file.' Thus the entire input parameter directoryName is a "path prefix".

Places to Verify

FileSystem and FileManager

IDL References

Data

```
enum ErrorNumberType {
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
    CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,
```

CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };

Exceptions

```
exception InvalidFileName {
        ErrorNumberType errorNumber;
        string msg; };
```

Operations

```
void rmdir ( in string directoryName )
    raises (CF::InvalidFileName, CF::FileException);
```

Preconditions

- The FileSystem and FileManager source code files are available.

Test Description

A. Identify the source code files that implement the FileSystem and FileManager *rmdir* operation. (OE0566, OE0599)
   1. **Untested**: The *rmdir* operation source code files are not available.

For each FileSystem and FileManager *rmdir* operation implementation found within the OE under test do the following steps:

B. Verify that the *rmdir* operation raises the *InvalidFileName* exception when the input directoryName parameter is not a valid path prefix. (OE0566).
   1. **Pass:** The *rmdir* operation raises the *InvalidFileName* exception when the input directoryName parameter is not a valid path prefix..
   2. **Fail:** The *rmdir* operation does not raise the *InvalidFileName* exception when the input directoryName parameter is not a valid path prefix.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)
   1. **Pass:** The *InvalidFileName* exception provides an error number with a CF::ErrorNumberType value for the error condition.
   2. **Fail:** The *InvalidFileName* exception does not provide an error number with a CF::ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_142				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the FileSystem and FileManager *rmdir* operation. (OE0566, OE0599)				
1. Perform a keyword search for *rmdir* operation on all FileSystem and FileManager source code provided by the developer. Examine the source code files returned and search for *rmdir* operation implementation. Record the file names.	The filenames are recorded. **Untested:** No results from keyword search. There is no implementation of the *rmdir* operation found.		May need the help of the software engineer to locate the source code files directories.	
For each FileSystem and FileManager *rmdir* operation implementation found within the OE under test do the following steps:				
B. Verify that the FileSystem *rmdir* operation raises the *InvalidFileName* exception when the input directoryName parameter is not a valid path prefix. (OE0566)				
2. Find where the input parameter directoryName is verified as a valid path prefix.	**Pass**: Verification of the input parameter directoryName as a valid path prefix is found. (OE0566) **Fail**: Verification of the input parameter directoryName as a valid path prefix is not found. (OE0566)		So one should look for the verification of the path prefix, like a call to the FileSystem with the input parameter string followed by a test for success. The FileSystem call could be *list* or *rmdir*. One side of the test should lead to the removal of the directory. The other side of the test should lead to raising an exception.	

OE_TC_142				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Determine that *InvalidFileName* exception is raised when the input directoryName parameter is not a valid path prefix.	**Pass:** The *rmdir* operation raises the *InvalidFileName* exception when the input directoryName parameter is not a valid path prefix. (OE0566) **Fail:** The *rmdir* operation does not raise the *InvalidFileName* exception when the input directoryName parameter is not a valid path prefix. (OE0566)			
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)				
4. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_142				
Step 1 (Source file name)	**Step 2** (Found verification of input parameter) Y/N	**Step 3** (Exception raised) Y/N	**Step 4** (Correct error number type) Y/N	**Notes**

Test Summary OE_TC_142

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

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

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

OE0566_____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.26. OE_TC_143 - FileManager :: mount raises InvalidFileName

Test Case Number: OE_TC_143

FileManager::mount raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0575	The *mount* operation shall raise the CF *InvalidFileName* exception when the input mount point does not conform to the file name syntax in section 3.1.3.4.2.1.
OE0599	The errorNumber parameter shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-88, Section 3.1.3.4.3.5.1.5 Page 3-93, Section 3.1.3.6.4 Page 3-79, Section 3.1.3.4.2.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 and C-4 ; C-12, C-26
SCA Appendix D : Domain Profile	Version 2.2.2 15 May 2006	Page D-53, Section D.7.1.7

Test Objective

This test case verifies OE0575 and OE0599. The objective of this test is to (1) verify that the *mount* operation raises the *InvalidFileName* exception when the input mount point does not conform to the file name syntax in section 3.1.3.4.2.1. , (2) verify that there is an appropriate error number of the type CF::ErrorNumberType as part of the raised exception.

Places to Verify

Core Framework FileManager *mount* operation

IDL References

Data

```
enum ErrorNumberType {
        CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
        CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
        CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,
```

CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV};

Exceptions

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

Operations

```
void mount (
        in string mountPoint,
        in CF::FileSystem file_System
        )
        raises( CF::InvalidFileName, CF::FileManager::InvalidFileSystem, CF::FileManager::MountPointAlreadyExists);
```

Preconditions

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

Test Description

A. Identify the FileManager's source code files that implement the *mount* operation. (OE0575)
   1. **Pass**: The source code file(s) of the *mount* operation are identified.
   2. **Fail**: The source code file(s) of the *mount* operation are not identified.

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

B. Verify that the *mount* operation raises the *CF::InvalidFileName* exception when the input mount point does not conform to the file name syntax in section 3.1.3.4.2.1. (OE0575)
   5. **Pass**: The *mount* operation raises the *CF::InvalidFileName* exception for the non-conforming file name syntax.
   6. **Fail:** The *mount* operation does not raise the *CF::InvalidFileName* exception for the non-conforming file name syntax.

C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0599)
   1. **Pass**: The exception includes an error number (errorNumber) of ErrorNumberType.
   2. **Fail**: The exception does not include an error number (errorNumber) of ErrorNumberType.

Manual Test Steps

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

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

OE_TC_143				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *mount* operation. (OE0575)				
1. Search for the implementation of the FileManager *mount* operation and record the file name.	**Pass**: The source code file(s) of the *mount* operation are identified. (OE0575) **Fail**: The source code file(s) of the *mount* operation are not identified. (OE0575) .		May need the help of the software engineer to locate the source code files.	
For each *mount* operation found within the OE under test, perform the following steps:				
B. Verify that the *mount* operation raises the *CF::InvalidFileName* exception when the input mount point does not conform to the file name syntax in section 3.1.3.4.2.1. (OE0575)				
2. Determine that the *CF::InvalidFileName* exception is raised when the input mount point file name does not conform to the following file name syntax: Valid individual filenames and directory names shall be 40 characters or less.	**Pass**: The *mount* operation raises the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575) **Fail:** The *mount* operation does not raise the *CF::InvalidFileName* exception for the non-conforming file name syntax. *(*OE0575)		(Steps within this section B reflect the different file name syntax requirements listed in the SCA's section 3.1.3.4.2.1. Each syntax requirement identified in this section B should be verified independent of the other file name syntax requirements.)	

OE_TC_143				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Determine that the *CF::InvalidFileName* exception is raised when the input mount point file name does not conform to the following file name syntax: Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters.	**Pass**: The *mount* operation raises the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575) **Fail:** The *mount* operation does not raise the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575)			
4. Determine that the *CF::InvalidFileName* exception is raised when the input mount point file name does not conform to the following file name syntax: The filenames "." ("dot") and ".." ("dot-dot") are invalid in the context of a file system.	**Pass**: The *mount* operation raises the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575) **Fail:** The *mount* operation does not raise the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575)			
5. Determine that the *CF::InvalidFileName* exception is raised when the input mount point file name does not conform to the following file name syntax: Valid pathnames are structured according to the POSIX specification whose valid characters include the "/" (forward slash) character in addition to the valid filename characters. (A valid pathname may consist of a single filename.)	**Pass**: The *mount* operation raises the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575) **Fail:** The *mount* operation does not raise the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575)			

OE_TC_143				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Determine that the *CF::InvalidFileName* exception is raised when the input mount point file name does not conform to the following file name syntax: A valid pathname shall not exceed 1024 characters.	**Pass**: The *mount* operation raises the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575) **Fail:** The *mount* operation does not raise the *CF::InvalidFileName* exception for the non-conforming file name syntax. (OE0575)			
C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0599)				
7. Determine if the exception includes an error number (*errorNumber*) of the type *ErrorNumberType*.	**Pass:** The exception includes an error number (*errorNumber*) of *ErrorNumberType*. (OE0599) **Fail:** The exception does not include an error number (*errorNumber*) of *ErrorNumberType*. (OE0599)			
End of Test				

Test Recording Log – OE_TC_143							
Step 1 (source file name)	**Step 2** (40 Char)	**Step 3** (Alpha Numeric)	**Step 4** (Dot)	**Step 5** (slash)	**Step 6** (1024 char)	**Step 7** (error number)	**Notes**

Test Summary OE_TC_143

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

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

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

OE0575_____

OE0599_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.27. OE_TC_144 - FileSystem :: copy raises InvalidFileName when inputs are the same

Test Case Number: OE_TC_144

FileSystem::copy raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0599	The error number shall indicate a CF ErrorNumberType value.
OE0735	The *copy* operation shall raise the CF InvalidFileName exception when the destination pathname is identical to the source pathname.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page,3-82 Section 3.1.3.4.2.5.2.5; Page 3-93, Sec. 3.1.3.6.4;
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3 and C-4 ; C-12, C-26

Test Objective

This test case verifies OE0735 and OE0599. The objective of this test is to verify that the *copy* operation throws the CF::InvalidFileName exception when the source and destination files are the same file and to verify the error number type accompanying the exception.

Places to Verify

FileSystems, FileManager

IDL References

Data

```
enum ErrorNumberType {
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
    CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,
    CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,
    CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE,
    CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };
```

Exceptions

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

Operations

void copy (
in string sourceFileName,
in string destinationFileName
)
raises (CF::InvalidFileName, CF::FileException);

Preconditions

- The *FileSystem* and *FileManager* source code files are available.

Test Description

A. Locate the source code for the *FileSystem copy* operation. (OE0735, OE0599)
   1. **Pass:** The source code for the *copy* operation is located.
   2. **Fail:** The source code for the copy operation is not located.

For each source file found in step A, perform steps B through D.

B. Verify that the InvalidFileName exception is raised when the source and destination files are the same file. (OE0735)
   1. **Pass:** The InvalidFileName exception is raised when the source and destination files are the same file.
   2. **Fail:** The InvalidFileName exception is not raised when the source and destination files are the same file.

C. Verify that there is a valid error number with the exception. (OE0599)
   1. **Pass:** The error number is valid.
   2. **Fail:** The error number is not valid.
   3. **Fail**: the is no error number with the exception

Manual Test Steps

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

OE_TC_144				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code for the *FileSystem copy* operation. (OE0735, OE0599)				
1. Locate all declarations of the *copy* operation	Declarations of *copy* operations are found. (OE0735, OE0599)		The intent of these two steps is to ensure we find the implementation of the copy operation for every device and to minimize the next steps by having to look at base class implementations of the copy operation only once.	
2. Determine which instances implement the *copy* operation and record the name of the file containing the implementation (source code file).	**Pass**: Implementations of the *copy* operations are found. (OE0735, OE0599) **Fail**: No implementations of the *copy* operation are found. (OE0735, OE0599)			
For each source file found in step A, perform steps B through D.				
B. Verify that the InvalidFileName exception is raised when the source and destination files are the same file. (OE0735)				
3. Determine that CF::InvalidFileName is raised when the source and destination files are the same file names.	**Pass:** The *copy* operation raises the CF::InvalidFileName when the source and destination files are the same file names. (OE0735) **Fail:** The *copy* operation does not raise the CF::InvalidFileName when the source and destination files are the same file names. (OE0735)			
C. Verify that there is a valid error number with the exception. (OE0599)				

OE_TC_144				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0599) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_144			
Step 2 (Implementations of *copy* operation)	Step 3 (exception raised)	Step 4 (Error number included)	Notes

Test Summary OE_TC_144

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

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

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

OE0599_____

OE0735 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.28. OE_TC_146 - File :: write raises IOException

Test Case Number: OE_TC_146

File::write raises IOException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0519	The *write* operation shall raise the *IOException* when a write error occurs.
OE0519-C111	If the file was opened using the *FileSystem::open* operation with an input read_Only parameter value of TRUE, writes to the file are considered to be in error.
OE0507	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-77, Section 3.1.3.4.1.5.2.5 Page 3-76, Section 3.1.3.4.1.3.1 Page 3-84, Section 3.1.3.4.2.5.6.2
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3, C-4 ; C-9

Test Objective

This test case verifies OE0519 with criteria OE0519-C111, and OE0507. The objective of this test is to (1) verify that the *write* operation raises the *IOException* exception when a write error occurs, (2) when a read_Only file is opened to write, this should cause a write error; therefore the File::write operation is expected to raise the IOException exception, 3) verify that the error number in the exception is of the type CF::ErrorNumberType.

Places to Verify

Core Framework's File interface

IDL References

Data

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

CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV};

Exceptions

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

Operations

void write (in CF::OctetSequence data)
raises (CF::File::IOException);

Preconditions

The File source code of the Core Framework is available.

Test Description

A. Identify the source code files that implement the *File::write* operation. (OE0519)
   1. **Untested:** The search result returns 0 (zero) files and the *write* operation is not found in any of the source files upon completion of search.

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

B. Verify that the *write* operation raises the *CF::IOException* when a write error occurs. (OE0519, OE0519-C111)
   1. Determine if the write operation raises the CF::IOException when any write error occurs during the write operation. (OE0519)
      a. **Pass:** When the *write* operation encounters any write error while trying to write, it should handle the write error by raising the CF::IOException.
      b. **Fail:** The *write* operation does not handle any write error encountered during the *write* operation.
   2. Determine if the **write** operation raises the CF::IOException when the write error is caused by the write operation writing to a file opened with an input read_Only value of TRUE. (OE0519-C111)
      a. **Pass:** The *write* operation handles the condition when a file was opened with read_Only by raising the IOException exception.
      b. **Fail:** The *write* operation does not raise the IOException exception when it encounters a write error trying to write to files that was opened with read_Only value.

C. Verify that the error number that accompanies the write-error exception is a *CF::ErrorNumberType* value. (OE0507)

1. **Pass**: The exception includes an error number (errorNumber) of ErrorNumberType.
2. **Fail**: The exception does not include an error number (errorNumber) of ErrorNumberType.

Manual Test Steps

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

OE_TC_146				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *write* operation. (OE0519)				
1. Perform a keyword search for the *write* operation on all source code provided by the developer.	**Untested:** The search result returns 0 (zero) files.		May need the help of the software engineer to locate the source code files.	
2. Examine the source code files returned in Step 1 and search for the *write* operation implementation. Record the file(s) name.	**Untested:** The *write* operation is not found in any of the source files upon completion of search. (OE0519).			
A.1. Determine if the write operation raises the CF::IOException when any write error occurs during the write operation. (OE0519)				
3. From the files identified in Step 2, search within the *write* operation and determine if the *CF::IOException* exception is raised for the condition when any write error occurs.	**Pass**: When the *write* operation encounters any write error while trying to write, it should handle the write error by raising the CF::IOException. (OE0519) **Fail:** The *write* operation does not raise the *CF::IOException* to handle write errors. (OE0519)			
A.2. Determine if the *write* operation raises the CF::IOException when the write error is caused by the *write* operation writing to a file opened with an input read_Only value of TRUE. (OE0519-C111)				

OE_TC_146				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. From the files identified in Step 2, search within the *write* operation and determine if the *CF::IOException* exception is raised for the condition when a write error occurs because the file opened has a read_Only value.	**Pass**: The write operation handles the condition when a file was opened with read_Only by raising the IOException exception. (OE0519-C111) **Fail:** The write operation does not raise the IOException exception when it encounters a write error trying to write to files that was opened with read_Only value. (OE0519-C111)			
B. Verify that the error number that accompanies the write-error exception is a *CF::ErrorNumberType* value. (OE0507)				
5. Determine if the exception includes an error number (*errorNumber*) of the type *ErrorNumberType*.	**Pass:** The exception includes an error number (*errorNumber*) of *ErrorNumberType*. (OE0507) **Fail:** The exception does not include an error number (*errorNumber*) of *ErrorNumberType*. (OE0507)			
End of Test				

Test Recording Log – OE_TC_146				
Step 2 (source file name)	**Step 3** (exception raised – Y/N)	**Step 4** (exception raised – Y/N)	**Step 5** (error number – Y/N)	**Notes**

Test Summary OE_TC_146

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

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

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

OE0519 _____

OE0519-C111 _____

OE0507 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.29. OE_TC_147 - File :: sizeOf raises FileException

Test Case Number: OE_TC_147

File::sizeOf raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0521	The *sizeOf* operation shall raise the *CF FileException* when a file-related error occurs (e.g., file does not exist anymore).
OE0598	The error number shall indicate *a CF ErrorNumberType* value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-78, Section 3.1.3.4.1.5.3.5 Page 3-93, Section 3.1.3.6.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-8, C-9

Test Objective

This test case verifies OE0521 and OE0598. The objective of this test is to (1) verify that the *FileException* is raised when a file related error occurs during the *sizeOf* operation, and (2) verify that there is an appropriate error number of *CF::ErrorNumberType* as part of the exception.

Places to Verify

File component

IDL References

Data

```
enum ErrorNumberType {
        CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
        CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
        CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,
        CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,
        CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS,
        CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };
```

Exceptions

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

Operation

unsigned long sizeOf () raises (CF::FileException);

Preconditions

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

Test Description

A. Identify the source code files that implement the *sizeOf* operation.
   1. **Untested**: The source code files are not available. (OE0521)

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

B. Verify that the *sizeOf* operation raises the *FileException* when a file related error occurs. (OE0521)
   1. **Pass:** The sizeOf operation raises FileException.
   2. **Fail:** The sizeOf operation does not raise FileException.

C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0598)
   1. **Pass:** The *FileException* exception provides an error number with an ErrorNumberType value for the error condition.
   2. **Fail:** The *FileException* exception does not provide an error number with an ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_147				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *sizeOf* operation. OE0521)				
1. Perform a search for the implementation of *sizeOf* operation on the source code provided by the developer. Record the file(s) name	**Untested:** The source code of the *sizeOf* operation is not available. (OE0521)		May need the help of the software engineer to locate the source code.	
For each *sizeOf* operation found within the OE under test, perform the following steps:				
B. Verify that the *sizeOf* operation raises the *FileException* when a file related error occurs. (OE0521)				
2. From the files identified in Step 1, search within the *sizeOf* operation and determine if the *CF::FileException* exception is raised for the condition when the target file no longer exists.	**Pass**: When the *sizeOf* operation encounters a file no longer exists error, it handles the file error by raising the *CF::FileException*. (OE0521) **Fail:** The *sizeOf* operation does not handle file related error such as file no longer exists with the *CF::FileException*. (OE0521)			
3. From the files identified in Step 1, search within the sizeOf operation and determine that *FileException* is raised when for the condition when any file-related error that may occur (other than file no longer exists).	**Pass**: The *sizeOf* operation raises the *FileException* to handle file related error. **(OE0521)** **Fail:** The *sizeOf* operation does not raise the *FileException* to handle file related error. (OE0521)			
C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0598)				

OE_TC_147				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0598)			
End of Test				

Test Recording Log – OE_TC_147			
Step 1 (source file name)	**Step 2 & 3** (exception raised – Y/N)	**Step 4** (error number – Y/N)	**Notes**

Test Summary OE_TC_147

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

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

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

OE0521_____

OE0598_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.30. OE_TC_148 - FileSystem::copy raises FileException

Test Case Number: OE_TC_148

FileSystem::copy raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0536	The copy operation shall raise the *CF FileException* exception when a file-related error occurs.
OE0598	The error number shall indicate *a CF ErrorNumberType* value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA) copy operation FileException	Version 2.2.2 15 May 2006	Page 3-82, Section 3.1.3.4.2.5.2.5 Page 3-93, Section 3.1.3.6.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	C-7

Test Objective

This test case verifies OE0536 and OE0598. The objective of this test is to (1) verify that the *FileException* is raised when a file related error occurs during the *copy* operation, (2) verify that the error number value in the exception is of the type *ErrorNumberType*.

Places to Verify

FileSystem and FileManager

IDL References

Data

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

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

Exceptions

exception FileException {ErrorNumberType errorNumber; string msg;};

Operations

void copy (in string sourceFileName, in string destinationFileName)
raises (CF::InvalidFileName, CF::FileException);

Preconditions

- The FileSystem and FileManager source code of the Core Framework is available.

Test Description

A. Identify the source code files that implement the *copy* operation.
   1. **Untested**: The source code files are not available. (OE0536)

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

B. Verify that the *copy* operation raises the *FileException* when a file related error occurs. (OE0536)
   1. **Pass:** The copy operation raises FileException.
   2. **Fail:** The copy operation does not raise FileException.

C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0598)
   1. **Pass:** The *FileException* exception provides an error number with an ErrorNumberType value for the error condition.
   2. **Fail:** The *FileException* exception does not provide an error number with an ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_148				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *copy* operation. (OE0536)				
1. Perform a search for the *copy* operation on FileSystem and FileManager source code provided by the developer.	**Untested:** The source code of the *copy* operation is not available. (OE0536)		May need the help of the software engineer to locate the source code.	
2. Examine the source code files returned in Step 1 and search for the *copy* operation implementation. Record the file name.	*copy* operation for each implementation is identified and recorded.			
For each *copy* operation found within the OE under test, perform the following steps:				
B. Verify that the *copy* operation raises the FileException when a file related error occurs. (OE0536)				
3. Search within the *copy* operation where a plain file is copied to another file.	Identified the source code where a plain file is copied to another file.			
4. Determine that *FileException* is raised when a file-related error occurs.	**Pass**: The *copy* operation raises the *FileException*. (OE0536) **Fail:** The *copy* operation does not raise the *FileException*. (OE0536)			
C. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0598)				

OE_TC_148				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
5. Determine if the exception includes an error number (errorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. **(**OE0598)			
End of Test				

Test Recording Log – OE_TC_148			
Step 2 (source file name)	**Step 4** (exception raised – Y/N)	**Step 5** (error number – Y/N)	**Notes**

Test Summary OE_TC_148

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

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

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

OE0536_____

OE0598_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.31. OE_TC_149 - FileSystem :: create raises FileException

Test Case Number: OE_TC_149

FileSystem::create raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0550	The *create* operation shall raise the *CF FileException* if the file already exists or another file error occurred.
OE0598	The error number shall indicate a *CF ErrorNumberType* value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-83, Section 3.1.3.4.2.5.5.5 Page 3-93, Section 3.1.3.6.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-4, C-7

Test Objective

This test case verifies OE0550 and OE0598. The objective of this test is three-fold (1) verify that the *FileException* is raised when the file already exists or another file error occurs during the File *create* operation, (2) verify that the error number value in the exception is of the type *ErrorNumberType*.

Places to Verify

FileSystem, FileManager

IDL References

Data

```
enum ErrorNumberType {
        CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
        CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
        CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,
        CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,
        CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS,
        CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };
```

Exceptions

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

Operations

File create (in string fileName) raises (InvalidFileName, FileException);

Preconditions

- The FileSystem and FileManager source code having the File *create* operation of the Core Framework are available.

Test Description

A. Identify the source code files that implement the File *create* operation. (OE0550)
   1. **Pass:** The File create operation is implemented in the source code.
   2. **Fail:** The File create operation is NOT implemented in the source code.

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

B. Verify that the File *create* operation raises the *FileException* if the file already exists. (OE0550)
   1. **Pass**: The File *create* operation raises *FileException* if the file already exists.
   2. **Fail:** The File *create* operation does not raise *FileException* if the file already exists.

C. **Verify** that the *File create* operation raises the *FileException* when another file error occurs. (OE0550)
   1. **Pass**: The File *create* operation raises *FileException* when another file error occurs.
   2. **Fail:** The File *create* operation does not raise *FileException* when another file error occurs.

D. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0598)
   1. **Pass:** The *FileException* exception provides an error number with an ErrorNumberType value for the error condition.
   2. **Fail:** The *FileException* exception does not provide an error number with an ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_149				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the File *create* operation. (OE0550)				
1. Examine the source code files of the file systems and file managers having the *create* operation. List the source code file name(s).	**Pass:** The File *create* operation is implemented in the source code. (OE0550) **Fail:** The File *create* operation is NOT implemented in the source code. (OE0550)		May need the help of the software engineer to locate the source code.	
2. Identify the location(s) where the *FileException* is raised by the *create* operation.	The File *create* operation for each implementation is identified and recorded.			
For each File *create* operation found within the OE under test, perform the following steps:				
B. Verify that the File *create* operation raises the *FileException* if the file already exists. (OE0550)				
3. Search within the File *create* operation where a new File is created.	**Pass**: The File *create* operation raises the *FileException* if the file already exists. (OE0550) **Fail:** The File *create* operation does not raise the *FileException* if the file already exists. (OE0550)			
C. Verify that the File *create* operation raises the *FileException* when another file error occurs. (OE0550)				

OE_TC_149				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Search within the File *create* operation where the FileException is raised for another file error.	**Pass**: The File *create* operation raises the *FileException* when another file error occurs. (OE0550) **Fail:** The File *create* operation does not raise the *FileException* when another file error occurs. (OE0550)			
D. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0598)				
5. Determine if the exception includes an error number (errorNumber) of the type; ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (OE0598)			
End of Test				

Test Recording Log – OE_TC_149				
Step 2 (source file with File *create* operation)	**Step 3** (*FileException* raised when file does not exist-Y/N?)	**Step 4** (*FileException* raised for other errors-Y/N?)	**Step 5** (error number-Y/N?)	**Notes**

Test Summary OE_TC_149

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

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

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

OE0550_____

OE0598_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.32. OE_TC_150 - FileSystem :: open raises FileException

Test Case Number: OE_TC_150

FileSystem::open raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0558	The *open* operation shall raise the CF *FileException* if the file does not exist or another file error occurred.
OE0598	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-84, Section 3.1.3.4.2.5.6.5 Page 3-93, Section 3.1.3.6.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	C-3 to C-4, C-8

Test Objective

This test case verifies OE0558 and OE0598. The objective of this test is to verify that the FileSystem and FileManager *open* operation raises the CF *FileException* exception if the file does not exist or another file error occurred. Furthermore, verify the exception contains an error number of the type CF::ErrorNumberType.

Places to Verify

FileSystem and FileManager

IDL References

Data

```
enum ErrorNumberType {
    CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED,
    CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR,
    CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE,
    CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR,
    CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE,
    CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };
```

Exceptions

```
exception FileException {
                CF::ErrorNumberType errorNumber;
                string msg;};
exception InvalidFileName { CF::ErrorNumberType errorNumber; string msg; };
```

Operations

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

Preconditions

- The FileSystem and FileManager source code files are available.

Test Description

A. Identify the source code files that implement the FileSystem *open* operation. (OE0558, OE0598)
   1. **Untested**: The *open* operation source code files are not available.

For each FileSystem and FileManager *open* operation implementation found within the OE under test do the following steps

B. Verify that the FileSystem *open* operation raises the *FileException* exception when the file does not exist or another file error occurred. (OE0558).
   1. **Pass:** The *open* operation raises the *FileException* exception when the file does not exist or another file error occurred.
   2. **Fail:** The *open* operation does not raise the *FileException* exception when the file does not exist or another file error occurred.

C. Verify that the error number that accompanies the exception is a CF ErrorNumberType value. (OE0598)
   1. **Pass:** The *FileException* exception provides an error number with a CF ErrorNumberType value for the error condition.
   2. **Fail:** The *FileException* exception does not provide an error number with a CF ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_150				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the FileSystem *open* operation. (OE0558, OE0598)				
1. Perform a keyword search for *open* operation on all FileSystem and FileManagers source code provided by the developer. Examine the source code files returned in Step 1 and search for *open* operation implementation. Record the file name.	The *open* operation for each implementation is identified and recorded. The file names are recorded. **Untested:** No results from keyword search. There is no implementation of the *open* operation found.		May need the help of the software engineer to locate the source code files directories.	
For each FileSystem and FileManager *open* operation implementation found within the OE under test do the following steps				
B. Verify that the FileSystem *open* operation raises the *FileException* exception when the file does not exist or another file error occurred. (OE0558)				
2. Find where the attempt to open the file object fails because the file object does not exist.	**Pass:** The code, where the attempt to open the file object fails because the file object does not exist, is found. (OE0558) **Fail:** The code, where the attempt to open the file object fails because the file object does not exist, is not found. (OE0558)			

OE_TC_150				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Determine that *FileException* exception is raised when the file does not exist.	**Pass:** The *open* operation raises the *FileException* exception when the file does not exist. (OE0558) **Fail:** The *open* operation does not raise the *FileException* exception when the file does not exist. (OE0558)		There are two reasons to get a FileException raised. This step verifies the one where the file does not exist.	
4. Find all places where an attempt to open the file object fails for a reason other than the input fileName is invalid.	**Pass:** Source code locations, where an attempt to open the file object fails for a reason other than the input fileName is invalid, are found. (OE0558) **N/A:** No source code locations, where an attempt to open the file object fails for a reason other than the input fileName is invalid, are found. (OE0558)		There are two reasons to get a FileException raised. This step verifies the one where the file does exist and some other file error occurs.	
Each location found in Step 4 represents "another file error", so for each location found				
5. Determine that *FileException* exception is raised when another file error occurred.	**Pass:** The *open* operation raises the *FileException* exception when another file error occurred. (OE0558) **Fail:** The *open* operation does not raise the *FileException* exception when another file error occurred. (OE0558)			
C. Verify that the error number that accompanies the exception is a CF ErrorNumberType value. (OE0598)				

OE_TC_150				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Determine if the exception includes an error number (*errorNumber*) of the type ErrorNumberType.	**Pass:** The exception includes an error number (*errorNumber*) of ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (*errorNumber*) of ErrorNumberType. (OE0598)			
End of Test				

Test Recording Log – OE_TC_150						
Step 1 (Source file names)	**Step 2** (File obj. doesn't exist) Y/N	**Step 3** (raise the proper exception) Y/N	**Step 4** (Found all other file errors) Y/N	**Step 5** (raise the proper exception) Y/N	**Step 6** (Correct error number type) Y/N	**Notes**

Test Summary OE_TC_150

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

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

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

OE0558_____

OE0598 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.33. OE_TC_151 - FileSystem :: mkdir raises FileException

Test Case Number: OE_TC_151

FileSystem::mkdir raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0562	The *mkdir* operation shall raise the CF *FileException* if the directory indicated by the input directoryName parameter already exists or if a file-related error occurred during the operation.
OE0598	The errorNumber parameter shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
SCA *mkdir* operation *FileException* exception	Version 2.2.2 15 May 2006	 Page 3-84, Section 3.1.3.4.2.5.7.5 Page 3-93, Section 3.1.3.6.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-3, C-4 and C-8

Test Objective

This test case verifies OE0562 and OE0598. The objective of this test is to (1) verify that the *mkdir* operation raises the *FileException* exception if the directory indicated by the input directoryName parameter already exists or if a file-related error occurred during the operation, (2) verify that there is an appropriate error number of type CF::ErrorNumberType as part of the raised exception.

Places to Verify

Core Framework FileSystem and FileManager

IDL References

Data

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

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

Exceptions

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

Operations

```
void mkdir (
    in string directoryName
    )
    raises (CF::InvalidFileName, CF::FileException);
```

Preconditions

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

Test Description

A. Identify the FileSystem's source code files that implement the *mkdir* operation. (OE0562)
   1. **Untested**: The FileSystem's source code files of the *mkdir* operation are not available.

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

B. Verify that the *mkdir* operation raises the *CF::FileException* exception if the directory indicated by the input *directoryName* parameter already exists or if a file-related error occurred during the operation. (OE0562)
   1. **Pass**: The *mkdir* operation raises *CF::FileException*.
   2. **Fail:** The *mkdir* operation does not raise *CF::FileException*.

C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0598)
   1. **Pass**: The exception includes an error number (errorNumber) of ErrorNumberType.
   2. **Fail**: The exception does not include an error number (errorNumber) of ErrorNumberType.

Manual Test Steps

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

OE_TC_151				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *mkdir* operation. (OE0562)				
1. Perform a keyword search for the *mkdir* operation on all FileSystem and FileManager source code provided by the developer.	**Untested:** The source code files of the *mkdir* operation are not available. (OE0562)		May need the help of the software engineer to locate the source code files.	
2. Examine the source code files returned in Step 1 and search for the implementation of the *mkdir* operation. Record the file names.	The *mkdir* operation for each implementation is identified and recorded.			
For each *mkdir* operation found within the OE under test, perform the following steps				
B. Verify that the *mkdir* operation raises the *CF::FileException* exception if the directory indicated by the input directoryName parameter already exists <u>or</u> if a file-related error occurred during the operation. (OE0562)				
3. Locate the portion of code within the mkdir operation for where the directory indicated by the input directoryName is checked to see if pre-exists. Verify that if the directory already exists, a FileException exception will be raised.	**Pass**: The *mkdir* operation raises the *CF::FileException* exception. (OE0562) **Fail:** The *mkdir* operation does not raise the *CF::FileException* exception. (OE0562)		(One of two core steps for the requirement.)	

OE_TC_151				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Trace through the mkdir operation code, and verify that each time a file-related error occured, a FileException exception would be raised.	**Pass**: The *mkdir* operation raises the *CF::FileException* exception. (OE0562) **Fail:** The *mkdir* operation does not raise the *CF::FileException* exception. (OE0562)		(One of two core steps for the requirement.) May need to search for the string 'FileException' and how it relates to the *mkdir* operation.	
C. Verify that the error number that accompanies the exception is a *CF::ErrorNumberType* value. (OE0598)				
5. Determine if the exception includes an error number (*errorNumber*) of the type *ErrorNumberType*.	**Pass:** The exception includes an error number (*errorNumber*) of *ErrorNumberType*. (OE0598) **Fail:** The exception does not include an error number (*errorNumber*) of *ErrorNumberType*. (OE0598)			
End of Test				

Test Recording Log – OE_TC_151				
Step 2 (source file name)	**Step 3** (exception raised – Y/N)	**Step 4** (exception raised – Y/N)	**Step 5** (error number – Y/N)	**Notes**

Test Summary OE_TC_151

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

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

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

OE0562_____

OE0598_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.34. OE_TC_152 - FileSystem :: rmdir raises FileException

Test Case Number: OE_TC_152

FileSystem::rmdir raises FileException

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0565	The *rmdir* operation shall raise the *CF FileException* when the directory identified by the input *directoryName* parameter does not exist, the directory contains files, or an error occurs which prohibits the directory from being deleted.
OE0598	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA) rmdir operation FileException	Version 2.2.2 15 May 2006	 Page 3-85, Section 3.1.3.4.2.5.8.5 Page 3-93, Section 3.1.3.6.3
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-3, C-4, C-8

Test Objective

This test case verifies OE0565 and OE0598. The objective of this test is to (1) verify that the *FileException* is raised during the *rmdir* operation when the directory identified by the input *directoryName* parameter does not exist, the directory contains files, or an error occurs which prohibits the directory from being deleted, (2) verify that the error number value in the exception is of the type *ErrorNumberType*.

Places to Verify

FileSystem and FileManager

IDL References

Data

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

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

Exceptions

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

Operation

void rmdir (in string directoryName) raises (InvalidFileName, FileException);

Preconditions

- The FileSystem and FileManager source code of the Core Framework is available.

Test Description

A. Identify the source code files that implement the *rmdir* operation.
   1. **Untested**: The source code files of the *rmdir* operation are not available. (OE0565)

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

B. Verify that the *rmdir* operation raises the *FileException* when the directory identified by the input *directoryName* does not exist. (OE0565)
   3. **Pass**: The *rmdir* operation raises *FileException* when the directory does not exist.
   4. **Fail:** The *rmdir* operation does not raise *FileException* when the directory does not exist.

C. Verify that the *rmdir* operation raises the *FileException* when the directory identified by the input *directoryName* contains files and is not empty. (OE0565)
   1. **Pass**: The *rmdir* operation raises *FileException* when the directory contains files and is not empty.
   2. **Fail:** The *rmdir* operation does not raise *FileException* when the directory contains files and is not empty.

D. Verify that the *rmdir* operation raises the *FileException* when an error occurs prohibiting the directory from being deleted. (OE0565)
   1. **Pass**: The *rmdir* operation raises *FileException* when an error occurs prohibiting the directory from being deleted.
   2. **Fail:** The *rmdir* operation does not raise *FileException* when an error occurs prohibiting the directory from being deleted.

E. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0598)
   1. **Pass:** The *FileException* exception provides an error number with an ErrorNumberType value for the error condition.
   2. **Fail:** The *FileException* exception does not provide an error number with an ErrorNumberType value for the error condition.

Manual Test Steps

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

OE_TC_152				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the source code files that implement the *rmdir* operation. (OE0565)				
1. Perform a search for the *rmdir* operation on FileSystem and FileManager source code provided by the developer.	**Untested:** The source code of the *mkdir* operation is not available. (OE0565)		May need the help of the software engineer to locate the source code.	
2. Examine the source code files returned in Step 1 and search for the implementation of the *rmdir* operation. Record the file name.	The *rmdir* operation for each implementation is identified and recorded.			
For each *rmdir* operation found within the OE under test, perform the following steps:				
B. Verify that the *rmdir* operation raises the *FileException* when the directory identified by the input *directoryName* does not exist. (OE0565)				
3. Verify that when the rmdir operation attempts to remove the directory, identified by the input parameter, and the directory does not exist then the FileException is raised.	**Pass**: The *rmdir* operation raises the *FileException* when the directory does not exist. (OE0565) **Fail:** The *rmdir* operation does not raise the *FileException* when the directory does not exist. (OE0565)			
C. Verify that the *rmdir* operation raises the *FileException* when the directory identified by the input *directoryName* contains files and is not empty. (OE0565)				
4. Verify that when the rmdir operation attempts to remove the directory, identified by the input parameter, and the directory contains files then the FileException is raised.	**Pass**: The *rmdir* operation raises the *FileException* when the directory contains files. (OE0565) **Fail:** The *rmdir* operation does not raise the *FileException* when the directory contains files. (OE0565)			

OE_TC_152				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
D. Verify that the *rmdir* operation raises the *FileException* when an error occurs prohibiting the directory from being deleted. (OE0565)				
5. Verify that when the rmdir operation attempts to remove the directory, identified by the input parameter, and an error occurs prohibiting the directory from being deleted then the FileException is raised.	**Pass**: The *rmdir* operation raises *FileException* when an error occurs prohibiting the directory from being deleted. (OE0565) **Fail:** The *rmdir* operation does not raise *FileException* when an error occurs prohibiting the directory from being deleted. (OE0565)			
E. Verify that the error number that accompanies the exception is an ErrorNumberType value. (OE0598)				
6. Verify if the exception includes an error number (errorNumber) of the type; ErrorNumberType.	**Pass:** The exception includes an error number (errorNumber) of ErrorNumberType. (OE0598) **Fail:** The exception does not include an error number (errorNumber) of ErrorNumberType. (**O**E0598)			
End of Test				

Test Recording Log – OE_TC_152					
Step2 (source files with *rmdir* operation)	**Step3** (source file with FileException when directory does not exist)	**Step4** (source file with FileException when directory contains files)	**Step5** (source file with FileException when error occurs prohibiting directory from being deleted)	**Step6** (error number)	**Notes**

Test Summary OE_TC_152

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

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

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

OE0565_____

OE0598_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.35. OE_TC_159 - FileSystem :: exists raises InvalidFileName

Test Case Number: OE_TC_159

FileSystem::exists raises InvalidFileName

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0540	The *exists* operation shall raise the CF InvalidFileName exception when input fileName parameter is not a valid absolute pathname.
OE0599	The error number shall indicate a CF ErrorNumberType value.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-82, Section 3.1.3.4.2.5.3.5; Page 3-93, Section 3.1.3.6.4; Page 1-2, Section 1.3.1.1; Page 3-79, Section 3.1.3.4.2.1
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-4, C-7, Section C.1

Test Objective

This test case verifies OE0540 and OE0599. The objective of this test is to (1) verify that the *exists* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname, (2) verify that the error number of the exception is of the type, CF::ErrorNumberType.

The validity of a pathname is determined by Section 3.1.3.4.2.1 of the SCA, which describes the POSIX specification for pathnames. According to this section, the POSIX specification describes the only allowable characters to be the "/"(forward slash) for the absolute pathname in addition to the following characters which are: the 62 alphanumeric characters, the "."(period), the underscore("_") and the hyphen("-"), with the exception that a filename cannot be made up entirely of a single or double dot. Also defined in Section 3.1.3.4.2.1 are the length limitations for a valid SCA pathname. A pathname must not exceed the length of 40 characters for an individual file or directory name and the length limit of 1024 characters for the entire pathname.

Places to Verify

FileSystem, FileManager

IDL References

Data

enum ErrorNumberType {
CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR, CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };

Exceptions

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

Operations

```
boolean exists ( in string fileName )
    raises (CF::InvalidFileName);
```

Preconditions

- The source code files of the file system and the file manager of the Core Framework are available.

Test Description

A. Identify the source code files that implement the FileSystem *exists* operation. (OE0540).
   1. **Untested**: There are no implementations of the *exists* operation in the Core Framework.

For each FileSystem *exists* operation in the Core Framework, perform the following steps:

B. Verify that the *exists* operation raises the *CF::InvalidFileName* exception when the input fileName parameter is not a valid absolute pathname. (OE0540)

Note: The aspects of a valid absolute pathname are starting with a forward slash, containing only SCA v2.2.2 defined valid characters, and adhering to the length limits defined by the SCA v2.2.2 document.

   1. **Pass**: The *exists* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname.

2. **Fail**: The *exists* operation does not raise the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname.

C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)

1. **Pass**: The CF::InvalidFileName exception provides an error number with a CF::ErrorNumberType value for the error condition.
2. **Fail**: The CF::InvalidFileName exception does not provide an error number with a CF::ErrorNumberType value for the error condition

Manual Test Steps

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

OE_TC_159				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Identify the implementations of the *exists* operation in the Core Framework (OE0540).				
1. Examine the source code of the files systems and the file manager of the Core Framework and identify the implementations of the *exists* operation. Lists the filename(s) of the *exists* operation.	**Fail**: There are no implementations of the *exists* operation in the Core Framework. (OE0540)			
For every implementation of the *exists* operation in the Core Framework, perform the following steps:				
B. Verify that the *exists* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname. (OE0540) **Note: The aspects of a valid absolute pathname are starting with a forward slash, containing only SCA v2.2.2 defined valid characters, and adhering to the length limits for pathnames defined by the SCA v2.2.2 document.**				
2. Examine the source code of the exists operation and identify the locations where the *CF::InvalidFileName* exception is raised.	The location(s) where the CF::InvalidFileName exception is raised in the exists operation is identified.			

OE_TC_159				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
3. Examine the source code of the *exists* operation and verify that the operation raises the *CF::InvalidFileName* exception if the input *fileName* parameter is not an absolute pathname, in other words, doesn't start with a forward slash. (See note in comments column).	**Pass**: The *exists* operation raises the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname. (OE0540) **Fail**: The *exists* operation does not raise the *CF::InvalidFileName* exception when the input *fileName* parameter is not a valid absolute pathname. (OE0540)		Pathnames are used in accordance with the POSIX specification definition and may reference either a plain file or a directory. An "absolute pathname" is a pathname which starts with a(forward slash) character – a "relative pathname does not have a leading "/" character.(Page 1-2) "Valid pathnames are structured according to the POSIX specification whose valid characters include the "/" (forward slash) character in addition to the valid filename characters. A valid pathname may consist of a single filename." (Page 3-79)	

OE_TC_159				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the *CF::InvalidFileName* exception is raised by the *exists* operation when the input *fileName* parameter does not contain valid characters, as defined by the SCA v2.2.2. See the reference in the comments column.	**Pass**: The *CF::InvalidFileName* exception is raised by the *exists* operation when the input *fileName* parameter contains invalid characters. (OE0540) **Fail**: The *CF::InvalidFileName* exception is not raised by the *exists* operation when the input *fileName* parameter contains invalid characters. (OE0540)		"Valid characters for a filename or directory name are the 62 alphanumeric characters (Upper, and lowercase letters and the numbers 0 to 9) in addition to the "." (period), "_" (underscore) and "-" (hyphen) characters." (SCA v2.2.2, Page 3-79, Section 3.1.3.4.2.1) Note: An invalid combination would be a single filename containing only a "." Or a ".." as a filename.	
5. Verify that the *CF::InvalidFileName* exception is raised by the *exists* operation when the input *fileName* parameter does not conform to the identified syntax requirement, stated as the following: "Valid individual filenames and directory names shall be 40 characters or less."	**Pass**: The *CF::InvalidFileName* exception is raised when the input *fileName* parameter does not conform to the identified file name syntax requirement. (OE0540) **Fail:** The *CF::InvalidFileName* exception is not raised when the input *fileName* parameter does not conform to the identified syntax requirement. (OE0540)			
6. Determine that the *CF::InvalidFileName* exception is raised by the *exists* operation when the input *fileName* parameter does not conform to the identified syntax requirement, stated as the following: "A valid pathname shall not exceed 1024 characters."	**Pass**: The exception is raised when the input *fileName* parameter does not conform to the identified syntax requirement. (OE0540) **Fail:** The exception is not raised when the input fileName parameter does not conform to the identified file name syntax requirement. (OE0540)			

OE_TC_159				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
C. Verify that the error number that accompanies the exception is a CF::ErrorNumberType value. (OE0599)				
7. Determine if the exception includes an error number (ErrorNumber) of the type ErrorNumberType.	**Pass:** The exception includes an error number (ErrorNumber) of ErrorNumberType. (OE0599) **Fail:** The exception does not include an error number (ErrorNumber) of ErrorNumberType. (OE0599)			
End of Test				

Test Recording Log – OE_TC_159						
Step 1 (source file name)	**Step 3** (slash)	**Step 4** (alpha numeric)	**Step 5** (40 char limit)	**Step 6** (1024 char limit)	**Step 7** (ErrorNumberType)	**Notes**

Test Summary OE_TC_159

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

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

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

OE0540 _____

OE0599 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.4.36. OE_TC_265 - File Attributes

Test Case Number: OE_TC_265

File attributes

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0509	The readonly fileName attribute shall contain the pathname used as the input fileName parameter of the FileSystem::create operation when the file was created.
OE0510	The readonly filePointer attribute shall contain the current file position.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-86, Section 3.1.3.4.1.4
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-9

Test Objective

This test verifies requirements OE0509 and OE0510. The objective of this test case is to confirm that the attributes of the File interface are implemented correctly. First, verify the fileName attribute contains the pathname used as the input fileName parameter of the FileSystem::create operation when the file was created. Secondly, confirm the readonly filePointer attribute contains the current file position.

Places to Verify

File, FileSystem, and FileManager implementations.

IDL References

Attributes

readonly attribute string fileName;
readonly attribute unsigned long filePointer;

Methods

CF::File FileSystem::create (in string fileName) raises (CF::InvalidFileName, CF::FileException);

Preconditions

- The File, FileSystem, and FileManager interface source code files are available.

Test Description

A. Locate all implementations of the File interface. (OE0509, OE0510)
   1. **Fail**: There is no implementation of the of the File interface.

B. Locate all implementations of the FileSystem interface. (OE0509)
   1. **Fail**: There is no implementation of the of the FileSystem interface.

For each occurrence of the File interface perform the following:

C. Verify the readonly fileName attribute contains the pathname passed to it from the FileSystem::create operation when the file was created. (OE0509)
   1. **Pass:** The fileName attribute contains the pathname passed to it from the FileSystem::create operation when the file was created.
   2. **Fail**: The fileName attribute does not contain the pathname passed to it from the FileSystem::create operation when the file was created.

D. Verify the readonly filePointer attribute contains the current file position. (OE0510)
   1. **Pass:** The readonly filePointer attribute contains the current file position.
   2. **Fail**: The readonly filePointer attribute does not contain the current file position.

For each occurrence of the FileSystem and FileManager interfaces perform the following:

E. Verify the FileSystem::create operation passes the correct pathname to the file that is created by the operation. (OE0509)
   1. **Pass:** The pathname information is passed to the File created by the FileSystem::create operation.
   2. **Fail**: The pathname information is passed to the File created by the FileSystem::create operation.

Manual Test Steps

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

OE_TC_265				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate all implementations of the File interface. (OE0509, OE0510)				
1. Record the names of the implementation files.	**Fail**: There is no implementation of the File interface. (OE0509, OE0510)			
B. Locate all implementations of the FileSystem interface. (OE0509)				
2. Record the names of the implementation files.	**Fail:** There is no implementation of the File interface. (OE0509)			
For each implementation of the File interface perform the following:				
C. Verify the readonly fileName attribute contains the pathname passed to it from the FileSystem::create operation when the file was created. (OE0509)				
3. Verify that the fileName attribute is properly initialized with the parameter containing the proper pathname on creation of the File.	**Pass:** The fileName attribute is assigned the correct pathname information during creation of the File. (OE0509) **Fail**: The fileName attribute is not assigned the correct pathname information during creation of the File. (OE0509)		This assignment will likely be located in the constructor or an initialization method.	

OE_TC_265				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the fileName attribute that was initialized during creation of the File is properly returned by accessing the fileName attribute.	**Pass:** The value returned by accessing the fileName attribute contains the correct pathname. (OE0509) **Fail**: The value returned by accessing the fileName attribute does not contain the correct pathname. (OE0509)			
D. Verify the readonly filePointer attribute contains the current file position. (OE0510)				
5. Verify the value returned by accessing the filePointer attribute contains the current file position.	**Pass:** The value returned by accessing the filePointer attribute contains the correct file position. (OE0510) **Fail**: The value returned by accessing the filePointer attribute does not contain the correct file position. (OE0510)		It is likely that most Operating Environments will delegate this task to the underlying file system. In which case expect to see a call to the underlying file system to determine the current file position. If the OE chooses to track this manually then it will be necessary to confirm that it is set to zero during creation of the File, and updated appropriately by the read, write, and setFilePointer operations.	
For each implementation of the FileSystem and FileManager interfaces perform the following:				
E. Verify the FileSystem::create operation passes the correct pathname to the file that is created by the operation. (OE0509)				

OE_TC_265				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
6. Verify the FileSystem::create operation passes its fileName input parameter to the constructor of the File interface object as an input argument.	**Pass:** The create operation passes its fileName input parameter to the constructor of the File interface object as an input argument. (OE0509) **Fail**: The create operation does not pass its fileName input parameter to the constructor of the File interface object as an input argument. (OE0509)			
End of Test				

Test Recording Log – OE_TC_265					
Step 1 (File implements file name)	**Step 2** (FileSystem implements file name)	**Step 3** (fileName initialized)	**Step 4** (fileName returned)	**Step 5** (filePointer returned)	**Step 6** (FileSystem initializes)

Test Summary OE_TC_265

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

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

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

OE0509_____

OE0510_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer: _____

Date Tested: _____

Witness: _____

B.4.37. OE_TC_275 - FileSystem :: query Input Parameter

Test Case Number: OE_TC_275

FileSystem::query

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0567	The *query* operation shall return file system information to the calling client based upon the given fileSystemProperties' ID.
OE0568	The FileSystem::*query* operation shall recognize and provide the designated return values for the following fileSystemProperties (section 3.1.3.4.2.3.2): 1. SIZE - an ID value of "SIZE" causes the query operation to return an unsigned long long containing the file system size (in octets). 2. AVAILABLE SPACE - an ID value of "AVAILABLE SPACE" causes the query operation to return an unsigned long long containing the available space on the file system (in octets).
OE0569	The *query* operation shall raise the *UnknownFileSystemProperties* exception when the given file system property is not recognized.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-85, Sections 3.1.3.4.2.5.9.3, Page 3-86, Sections 3.1.3.4.2.5.9.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Pages C-5, C-8, Section C.1

Test Objective

This test case verifies OE0567, OE568 and OE569. The objective of this test is to verify that the *query* operation returns file system information based on the given fileSystemProperties' ID. It verifies that the *query* operation recognizes the ID values of SIZE and AVAILABLE SPACE and returns the corresponding information. Finally, it verifies that when an input file system property is not recognized, then an *UnknownFileSystemProperties* exception is raised.

Note: Normally FileSystem operations are inherited by the FileManager and this test case would ask that both interfaces be verified. However, FileManager has a *query* operation that differs from the FileSystems. Therefore, there is no need to verify these requirements in the FileManager.

Places to Verify

FileSystem interface

IDL References

Data

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

Exceptions

```
exception UnknownFileSystemProperties { properties invalidProperties; };
```

Operations

```
void query (inout Properties fileSystemProperties)
      raises (UnknownFileSystemProperties);
```

Preconditions

- The FileSystem source code files are available.

Test Description

A. Locate with the FileSystem source code all implementations of the *query* operation. (OE0567, OE0568, OE0569)
   1. **Fail**: There are no implementations of the *query* operation in the FileSystem interface.

For each implementation of the *query* operation in a FileSystem interface:

B. Verify that if a fileSystemProperties‘ ID is “SIZE”, the *query* operation returns an unsigned long long containing the file system size. (OE0568)
   1. **Pass:** If a fileSystemProperties‘ ID is “SIZE”, the *query* operation returns an unsigned long long containing the file system size.
   2. **Fail:** If a fileSystemProperties‘ ID is “SIZE”, the *query* operation does not return an unsigned long long.
   3. **Fail:** If a fileSystemProperties‘ ID is “SIZE”, the *query* operation does not return the file system size.

C. Verify that if a fileSystemProperties’ ID is “AVAILABLE SPACE”, the *query* operation return an unsigned long long containing the available space on the file system. (OE0568)
   1. **Pass:** If a fileSystemProperties’ ID is “AVAILABLE SPACE”, the *query* operation return an unsigned long long containing the available space on the file system (in octets).
   2. **Fail:** If a fileSystemProperties’ ID is “AVAILABLE SPACE”, the *query* operation does not return the available space on the file system.

D. Verify that the *query* operation raise the *UnknownFileSystemProperties* exception when the given file system property is not recognized. (OE0569)
   1. **Pass:** When the input file system property is not recognized, then the *query* operation raises the *UnknownFileSystemProperties* exception.
   2. **Fail:** When the input file system property is not recognized, and the *query* operation does not raise the *UnknownFileSystemProperties* exception.

E. Verify that the *query* operation returns file system information to the calling client based upon the given fileSystemProperties' ID. (OE0567)
   1. **Pass:** The *query* operation returns file system information, based upon the input fileSystemProperties' ID.
   2. **Fail:** The *query* operation does not return file system information based upon the input fileSystemProperties' ID.

Manual Test Steps

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

OE_TC_275				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate with the FileSystem source code all implementations of the *query* operation. (OE0567, OE0568, OE0569)				
1. Locate all implementations of the *query* operation within the FileSystem interface in the source code.	**Fail**: No *query* operation implementation is found. (OE0567, OE0568, OE0569) **Pass**: *Query* operation implementations are found. (OE0567, OE0568, OE0569)			
For each implementation of the *query* operation FileSystem interface:				
B. Verify that if a fileSystemProperties' ID is "SIZE", the *query* operation returns an unsigned long long containing the file system size. (OE0568)				
2. Verify that in the processing of the fileSystemProperties sequence, SIZE is one of the valid properties IDs.	**Pass**: When processing the fileSystemProperties sequence, SIZE is one of the valid property IDs. (OE0568) **Fail**: When processing the fileSystemProperties sequence, SIZE is not one of the valid property IDs. (OE0568)			
3. Verify that the returned property value for SIZE is an unsigned long long.	**Pass**: The returned value for the property with an ID of SIZE is an unsigned long long. (OE0568) **Fail**: The returned property value for SIZE is not an unsigned long long. (OE0568)			

OE_TC_275				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the returned value for the property with an ID of SIZE contains the file system size (in octets).	**Pass**: The returned value for the property with an ID of SIZE contains the file system size (in octets). (OE0568) **Fail**: The returned value for the property with an ID of SIZE does not contain the file system size (in octets). (OE0568)			
C. Verify that if a fileSystemProperties' ID is "AVAILABLE SPACE", the *query* operation return an unsigned long long containing the available space on the file system. (OE0568)				
5. Verify that in the processing of the fileSystemProperties sequence, AVAILABLE SPACE, is one of the valid properties IDs.	**Pass**: When processing the fileSystemProperties sequence, AVAILABLE SPACE, is one of the valid property IDs. (OE0568) **Fail**: When processing the fileSystemProperties sequence, AVAILABLE SPACE, is not one of the valid property IDs. (OE0568)			
6. Verify that the returned property value for AVAILABLE SPACE is an unsigned long long.	**Pass**: The returned value for the property with an ID of AVAILABLE SPACE is an unsigned long long. (OE0568) **Fail**: The returned property value for AVAILABLE SPACE is not an unsigned long long. (OE0568)			

OE_TC_275				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Verify that the returned value for the property with an ID of AVAILABLE SPACE contains the available space on the file system (in octets).	**Pass**: The returned value for the property with an ID of AVAILABLE SPACE contains the available space on the file system (in octets). (OE0568) **Fail**: The returned value for the property with an ID of AVAILABLE SPACE does not contain the available space on the file system (in octets). (OE0568)			
D. Verify that the *query* operation raise the *UnknownFileSystemProperties* exception when the given file system property is not recognized. (OE0569)				
8. Locate in the processing of the fileSystemProperties sequence, when one of the property IDs is not recognized (i.e. it is unexpected).	**Pass**; In the processing of the fileSystemProperties sequence there is processing for unexpected property IDs. (OE0569) **Fail**; In the processing of the fileSystemProperties sequence there is no processing for unexpected property IDs. (OE0569)			
9. Verify that when one of the property IDs is not recognized then the *UnknownFileSystemProperties* exception is raised.	**Pass**: Within the processing for unexpected property IDs, the *UnknownFileSystemProperties* exception is raised. (OE0569) **Fail**: Within the processing for unexpected property IDs, the *UnknownFileSystemProperties* exception is not raised. (OE0569)			
E. Verify that the *query* operation returns file system information to the calling client based upon the given fileSystemProperties' ID. (OE0567)				

OE_TC_275				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
10. Verify that the fileSystemProperties sequence is the bases for compiling file system information and returning it to the caller.	**Pass**: The fileSystemProperties sequence is the bases for compiling file system information and returning it to the caller. (OE0567) **Fail**: The fileSystemProperties sequence is not the bases for compiling file system information and returning it to the caller. (OE0567) **Fail**: The fileSystemProperties sequence is the bases for compiling file system information but that information is not returned to the caller. (OE0567)			
End of Test				

Test Recording Log – OE_TC_275				
Step 1 (Locate *query* operation)	**Step 2 - 4** (SIZE is a valid property)	**Step 5 - 7** (AVAILABLE SPACE is a valid property)	**Step 8 & 9** (bad property ID raises exception)	**Step 10** (property seq is basis for returned info)

Test Summary OE_TC_275

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

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

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

OE0567_____

OE0568 _____

OE0569 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.38. OE_TC_276 - FileManager :: mount

Test Case Number: OE_TC_276

FileManager::mount

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0573	The *mount* operation shall associate the specified file system with the mount point referenced by the input *mountPoint* parameter.
OE0574	A mount point name shall begin with a "/" (forward slash character).
OE0576	The *mount* operation shall raise the MountPointAlreadyExists exception when the mount point already exists in the file manager.
OE0577	The *mount* operation shall raise the InvalidFileSystem exception when the input FileSystem is a null object reference.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Page 3-87, Section 3.1.3.4.3.3.4 Page 3-88, Sections 3.1.3.4.3.3.4 , 3.1.3.4.3.5.1.3, and 3.1.3.4.3.5.1.5
SCA Appendix C: Core Framework IDL	Version 2.2.2 15 May 2006	Page C-12

Test Objective

This test case verifies OE0573, OE0574, OE0576 and OE0577. The objective of this test case is to verify that the *mount* operation validates the input mountPoint parameter, and verifies that it does not already exist. Furthermore, the test case verifies that the *mount* operation raises an exception if the mount point does already exist, and it raises an exception if the input file system object reference is null. Finally, the test case verifies that the *mount* operation associates the specified file system with the mount point referenced by the input parameter.

Places to Verify

FileManager interfaces

IDL References

Data

enum ErrorNumberType {

CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR, CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV };

Exceptions

```
exception InvalidFileName  { CF::ErrorNumberType  errorNumber;
                             string msg; };
exception InvalidFileSystem  { };
exception MountPointAlreadyExists  { };
```

Operations

```
void mount ( in string mountPoint,
             in CF::FileSystem  file_System  )
     raises (CF::InvalidFileName,  CF::FileManager::InvalidFileSystem,  CF::FileManager::MountPointAlreadyExists);
```

Preconditions

- The FileManager source code files are available.

Test Description

For each implementation of the FileManager *mount* operation:

A. Verify that the *mount* operation validates the input mount point parameter to determine whether it begins with a "/" (forward slash character). (OE0574)
   1. **Pass:** The *mount* operation verifies that the mount point parameter begins with a (/) forward slash character.
   2. **Fail:** The *mount* operation does not verify the mount point parameter begins with a (/) forward slash character.

B. Verify that the *mount* operation validates the input mount point parameter and raises the *MountPointAlreadyExists* exception if it already exists in the file manager. (OE0576)
   1. **Pass:** The *mount* operation raises the *MountPointAlreadyExists* exception when the input mount point parameter already exists in the file manager.
   2. **Fail:** The *mount* operation does not raise the *MountPointAlreadyExists* exception when the input mount point parameter already exists in the file manager.

C. Verify that the *mount* operation validates the input FileSystem parameter and raises the *InvalidFileSystem* exception when the input FileSystem is a null object reference. (OE0577)
   1. **Pass:** The *mount* operation raises the *InvalidFileSystem* exception when the input FileSystem parameter is a null object reference.
   2. **Fail:** The *mount* operation does not raise the *InvalidFileSystem* exception when the input FileSystem parameter is a null object reference.

D. Verify that the *mount* operation associates the file system specified with the input *mountPoint* parameter. (OE0573)
   1. **Pass:** The *mount* operation associates the file system specified with the input *mountPoint* parameter.
   2. **Fail:** The *mount* operation does not associate the file system specified with the input *mountPoint* parameter.

Manual Test Steps

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

OE_TC_276				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
For each implementation of the FileManager *mount* operation:				
A. Verify that the *mount* operation validates the input mount point parameter to determine whether it begins with a "/" (forward slash character). (OE0574)				
1. Find in the source code of the *mount* operation, where the validation of the *mountPoint* parameter occurs.	**Pass**: The source code of the *mount* operation, where the *mountPoint* parameter is validated, is found. (OE0574) **Fail**: The source code of the *mount* operation, where the *mountPoint* parameter is validated, is not found. (OE0574)			
2. Verify that the first character of the mount point parameter must be a forward slash character (/).	**Pass** The first character of the mount point parameter must be a forward slash character(/). (OE0574) **Fail**: There is no validation that the first character of the mount point parameter has to be a forward slash character(/). (OE0574)			
B. Verify that the *mount* operation validates the input mount point parameter and raises the *MountPointAlreadyExists* exception if it already exists in the file manager. (OE0576)				
3. Find where the mount point is validated for existence.	**Pass**: The place where the mount point is validated for existence is found. (OE0576) **Fail**: The place where the mount point is validated for existence is not found. (OE0576)			

OE_TC_276				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
4. Verify that the *MountPointAlreadyExists* exception is raised when it is determined that the mount point already exists.	**Pass**: The *MountPointAlreadyExists* exception is raised when it is determined that the mount point already exists. (OE0576) **Fail**: The *MountPointAlreadyExists* exception is not raised when it is determined that the mount point already exists. (OE0576)			
C. Verify that the *mount* operation validates the input FileSystem parameter and raises the *InvalidFileSystem* exception when the input FileSystem is a null object reference. (OE0577)				
5. Find where the FileSystem is validated for existence	**Pass**: The place where the FileSystem is validated for null object reference is found. (OE0577) **Fail**: The place where the FileSystem is validated for null object reference is not found. (OE0577)			
6. Verify that the *InvalidFileSystem* exception is raised when FileSystem is a null object reference.	**Pass**: The *InvalidFileSystem* exception is raised when FileSystem is a null object reference. (OE0577) **Fail**: The *InvalidFileSystem* exception is not raised when FileSystem is a null object reference. (OE0577)			
D. Verify that the *mount* operation associates the file system specified with the input *mountPoint* parameter. (OE0573)				

OE_TC_276				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Verify that the *mount* operation associates the input file system and with the mount point referenced by the input mountPoint parameter	**Pass**: The *mount* operation associates the input file system and with the mount point referenced by the input mountPoint parameter. (OE0573) **Fail**: the *mount* operation does not associate the input file system and with the mount point referenced by the input mountPoint parameter. (OE0573)			
End of Test				

Test Recording Log – OE_TC_276				
Step1&2 (validation of mountPoint slash)	**Step3&4** (raising MountPointAlreadyExists)	**Step5** (validation of FileSystem)	**Step6** (raising InvalidFileSystem)	**Step7** (associate file system with the mount point)

Test Summary OE_TC_276

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

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

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

OE0573 _____

OE0574 _____

OE0576 _____

OE0577 _____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

B.4.39. OE_TC_283 - FileSystem create Responsibilities

Test Case Number: OE_TC_283

DeviceManager's FileSystem create responsibilities

Requirements

SCA v2.2.2 Tag	SCA v2.2.2 Text
OE0475	The device manager shall create FileSystem components implementing the FileSystem interface for each OS file system.
OE0476	If multiple file systems are to be created, the device manager shall mount created file systems to a FileManager component (widened to a FileSystem through the FileSys attribute).
OE0476-C109	The mount points used for the created file systems are identical to the values identified in the filesystemnames element of the device manager's Device Configuration Descriptor.
OE0760	Each mounted file system name shall be unique within the device manager.

References

Document Name	Version/Date	Location (Pages, Section)
Software Communication Architecture (SCA)	Version 2.2.2 15 May 2006	Pages 3-52, Section 3.1.3.2.4.5
SCA Appendix D: Domain Profile	Version 2.2.2 15 May 2006	Pages D-54, Section D.7.1.7

Test Objective

This test case verifies OE0475, OE0476, OE0476-C109 and OE0760. The objective of this test case is to verify that the deviceManager creates a FileSystem component for each OS file system. It also verifies that

- the deviceManager is responsible for mounting multiple file systems if present to a FileManager component,
- the mount points used for the created files systems match those found in the DCD file, and
- each mounted file system has a unique name within the device manager.

Places to Verify

DeviceManager, FileSystem and FileManager

XML References

DomainProfile

<!ELEMENT filesystemnames

```
        ( filesystemname+ )>
<!ELEMENT filesystemname  EMPTY>
<!ATTLIST filesystemname
        mountname  CDATA        #REQUIRED
        deviceid  CDATA         #REQUIRED>
```

Preconditions

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

Test Description

A. Locate the source code (DeviceManager, FileSystem, and FileManager) that pertains to the DeviceManager (s) under test. (OE0475, OE0476)

1. **Untested**: The full device manager implementation is not provided.

For each DeviceManager, examine the provided product code (DeviceManager & supporting utility files) for the implementation that supports the creation of the Device Manager's FileSystem(s).

B. Verify that the DeviceManager creates FileSystem components based on the number of file systems in the OS. (OE0475)

1. **Pass:** The DeviceManager creates one or more FileSystems as needed by the OS.
2. **Fail:** The DeviceManager ignores the OS file systems (i.e. the file systems needed) as it creates the FileSystems.

C. If multiple file systems are to be created, verify that the device manager mounts the created file systems to a FileManager component.

1. Verify that there is a FileManager and its FileSystem is saved as the fileSys attribute of the deviceManager. (OE0476)
   a. **Pass:** When multiple file systems are to be created, the DeviceManager creates a FileManager, which is saved in the DeviceManager fileSys attribute.
   b. **Fail:** When multiple file systems are to be created, the DeviceManager fails to create a FileManager.
   c. **Fail:** When multiple file systems are to be created, the DeviceManager creates a FileManager but does not save it in the DeviceManager fileSys attribute.
2. Verify that the deviceManager mounts the created file systems to its FileManager component. (OE0476)
   a. **Pass:** When multiple file systems are to be created, the DeviceManager creates the FileSystems and mounts them to the DeviceManager's FileManager.
   b. **Fail:** When multiple file systems are to be created, the DeviceManager fails to create the FileSystems.

c. **Fail:** When multiple file systems are to be created, the DeviceManager creates the FileSystems but fails to mount them to the DeviceManager's FileManager.

3. Verify that the mount points for the created file systems match the values provided in the filesystemnames element of the device manager's DCD file. (OE0476, OE0476-C109)
   a. **Pass:** When multiple file systems are to be created, the DeviceManager mount points for the created file systems match the values provided in the filesystemnames element of the device manager's DCD file.
   b. **Fail:** When multiple file systems are to be created, the DeviceManager mount points for the created file systems do not match the values provided in the filesystemnames element of the device manager's DCD file.
   c. **Fail:** When multiple file systems are to be created, the DeviceManager mount points for the created file systems are never compared to the values provided in the filesystemnames element of the device manager's DCD file.
4. Verify that within the DeviceManager each mounted file system name is unique. (OE0760)
   a. **Pass:** Within the DeviceManager, each mounted file system name is unique.
   b. **Fail:** .Within the DeviceManager, each mounted file system names are not unique…

Note: The creation of the FileSystem or FileManager could be static or dynamic. Static in the sense that a predefined number of instances is predetermine and defined in the source code. It could also be dynamic in the sense that the DeviceManager could create an instance/instances via the standard DCD execution process.

Manual Test Steps

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

OE_TC_283				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
A. Locate the source code (DeviceManager, FileSystem, and FileManager) that pertains to the DeviceManager (s) under test. (OE0475, OE0476)				
1. Locate the source code DeviceManager that pertains to the DeviceManager (s) under test.	**Untested:** The full device manager implementation is not provided. (OE0475, OE0476)			
2. Locate where the DeviceManager creates FileSystem components.	**Untested:** The full device manager implementation is not provided. (OE0475, OE0476)			
For each DeviceManager, examine the provided product code (DeviceManager & supporting utility files) for the implementation that supports the creation of the Device Manager's FileSystem(s).				
B. Verify that the DeviceManager creates FileSystem components based on the number of file systems in the OS. (OE0475)				
3. Locate where the DeviceManager determines how many FileSystem components it needs to create.	**Pass**: The DeviceManager determines the number of FileSystem components that are needed. (OE0475) **Fail**: The DeviceManager does not determine the number of FileSystem components that are needed. (OE0475)		Number of OS file systems may be static (a hard coded value) or dynamic (some software determination of the proper count).	
4. Verify that the DeviceManager creates FileSystem components based on the number of file systems in the OS.	**Pass**: The DeviceManager uses the OS file systems to determine the number of FileSystem components to create. (OE0475) **Fail**: The DeviceManager does not use the OS file systems to determine the number of FileSystem components to create. (OE0475)			
C. If multiple file systems are to be created, verify that the device manager mounts the created file systems to a FileManager component.				

OE_TC_283				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
1. Verify that there is a FileManager and its FileSystem is saved as the fileSys attribute of the deviceManager. (OE0476)				
5. Verify that when there are multiple file systems there is a FileManager and its FileSystem is saved as the fileSys attribute of the deviceManager.	**Pass:** When multiple file systems are to be created, the DeviceManager creates a FileManager, which is saved in the DeviceManager fileSys attribute. (OE0476) **Fail:** When multiple file systems are to be created, the DeviceManager fails to create a FileManager. (OE0476) **Fail:** When multiple file systems are to be created, the DeviceManager creates a FileManager but does not save it in the DeviceManager fileSys attribute. (OE0476)			
2. Verify that the deviceManager mounts the created file systems to its FileManager component. (OE0476)				
6. Verify that when there are multiple file systems, the deviceManager mounts the created file systems to its FileManager component.	**Pass:** When multiple file systems have been created, the DeviceManager mounts the FileSystems to the DeviceManager's FileManager. (OE0476) **Fail:** When multiple file systems have been created, the DeviceManager fails to mount the FileSystems to the DeviceManager's FileManager. (OE0476)			
3. Verify that the mount points for the created file systems match the values provided in the filesystemnames element of the device manager's DCD file. (OE0476-C109)				

OE_TC_283				
Steps	**Expected Results**	**Actual Results**	**Comments**	**Test Result**
7. Verify that within the processing to mount each file system, the mount point for the file system matches the value provided in the filesystemnames element of the device manager's DCD file.	**Pass:** Within the processing to mount each file system, the DeviceManager's mount point for it matches the value provided in the filesystemnames element of the device manager's DCD file. (OE0476-C109) **Fail:** Within the processing to mount each file system, the DeviceManager's mount point for it does not match the value provided in the filesystemnames element of the device manager's DCD file. (OE0476-C109) **Fail:** Within the processing to mount each file system, the DeviceManager's mount point for it is never compared to the value provided in the filesystemnames element of the device manager's DCD file. (OE0476-C109)		Failure of criterion OE0476-C109 results in failure of the OE0476 requirement.	
4. Verify that within the DeviceManager each mounted file system name is unique. (OE0760)				
8. Verify that within the processing to mount each file system, its name is checked for uniqueness.	**Pass:** Within the processing to mount each file system, the file system's name is checked for uniqueness. (OE0760) **Fail:** Within the processing to mount each file system, the file system's name is not checked for uniqueness. (OE0760)			
End of Test				

Test Recording Log – OE_TC_283				
Step 1 & 2 (locate source code)				
Step 3 & 4 (FileSystems created is based on the number of OS file systems)	**Step 5** (FileManager is saved as DeviceManager's fileSys attribute.)	**Step 6** (file systems are mounted to FileManager)	**Step 7** (mount points match values listed in DCD)	**Step 8** (unique mounted file system names)

Test Summary OE_TC_283

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

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

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

OE0475_____

OE0476_____

OE0476-C109 _____

OE0760_____

Failed Items (Section/Step Number):

_____

_____

_____

Test Engineer:_____

Date Tested: _____

Witness: _____

Index of Test Case Titles for Manual Tests

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

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

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

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

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

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