# *Lighting Service Framework Controller Service 14.06 Test Case Specifications*

***January 5, 2015***

# Contents

# 1 Introduction

## 1.1 Purpose

These test cases evaluate and verify the functionality of an implementation of the AllJoyn Lighting Service Framework exposed by a device through the Controller Service's collection of interfaces.

**NOTE**

Refer to the AllJoyn Lighting Service Framework Controller Service Interface Definition document to verify your application includes all mandatory information necessary to meet the compliance and certification requirements.

## 1.2 Scope

These test cases are designed to determine if a device conforms to the Controller Service 14.06 interface specifications. Successful completion of all test cases does not guarantee that the tested device will interoperate with other devices.

This document is intended for software engineers and assumes familiarity with the AllJoyn SDK.

## 1.3 References

The following are reference documents

- *AllJoyn Lighting Service Framework Controller Service 14.06 Interface Definition*
- *AllJoyn Lighting Service Framework Lamp Service 14.06 Interface Definition*
- *AllJoyn About Feature 1.0 Interface Definition*

# 2 Environment Setup

## 2.1 Requirements

The following are required in order to execute these test cases:

- An AllJoyn-enabled device (the device under test or DUT) that implements the Controller Service interfaces according to the Lighting Service Framework Controller Service 14.06 specification.
- An AllJoyn-enabled device that implements the Lamp Service interfaces according to the Lighting Service Framework Lamp Service 14.06 specification. For example, a Lamp Service simulator, Luminaire available on the Google Play store will satisfy this requirement.
- A supported test device on which the test cases will run.
- A Wi-Fi access point (referred to as the personal AP).

## 2.2 Preconditions

Before running these test cases, it is assumed that:

- The DUT is connected to the personal AP
- The device running Lamp Service is connected to the personal AP
- The test device is connected to the personal AP
- At least once process on the DUT is announcing its capabilities through its About announcement, including its support for the Controller Service interfaces.

## 2.3 Parameters

**Table 1. Parameters for the Controller service tests**

| Parameter | Description |
|---|---|
| DeviceId | Device ID of the DUT |
| AppId | Application ID of the system application on the DUT (application implementing the Controller Service interface) |

# 3 Controller Service Test Cases

## 3.1 LSF_Controller-v1-01: Service interface XML matches

**Objective**

Verify that using introspection to retrieve each interface definition on the bus object matches exactly the XML maintained by the test suite. A successful match means that no interfaces have been added nor removed.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test compares the introspected interface definition with its stored XML file.
4. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The introspected interface definitions matches the Lamp Service XML maintained with the validation test.

## 3.2 LSF_Controller-v1-02: Service interface versions

**Objective**

Verify that getting the version property for each interface exposed by the Controller Service does not result in a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device calls getVersion() on the ControllerService bus object.
4. The test device calls getVersion() on the Lamp bus object.

5. The test device calls getVersion() on the Preset bus object.
6. The test device calls getVersion() on the LampGroup bus object.
7. The test device calls getVersion() on the Scene bus object.
8. The test device calls getVersion() on the MasterScene bus object.
9. The test device calls GetControllerServiceVersion() on the ControllerService bus object.
10. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- Each call to getVersion() does not result in a BusException on any interface.
- The call to GetControllerServiceVersion() does not result in a BusException.

## 3.3 LSF_Controller-v1-03: Reset controller service

**Objective**

Verify that calling the LightingResetControllerService() method will not result in a BusException and send out the appropriate signal.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device calls LightingResetControllerService() on the bus object.
4. The test device checks to see if it has received a LightingReset signal from the DUT.
5. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The method call to LightingResetControllerService() does not result in a BusException.

- The test device receives a LightingReset signal after calling LightingResetControllerService().

# 3.4 LSF_Controller-v1-04: Lamp info

**Objective**

Verify that calling the GetLampSupportedLanguages() and GetLampManufacturer() methods will not result in a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device calls GeLampSupportedLanguages() on the bus object.
5. The test device calls GetLampManufacturer() on the bus object.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call to GeLampSupportedLanguages() does not result in a BusException.
- The method call to GetLampManufacturer() does not result in a BusException.

# 3.5 LSF_Controller-v1-05: Get and set lamp name

**Objective**

Verify that the DUT can call GetLampName() and SetLampName() properly. GetLampName() should fetch the updated name after a call to SetLampName(). Furthermore, verify that the appropriate signal is sent out after the lamp name is changed.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.

2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device calls SetLampName() on the bus object to set the name of the connected lamp as “ControllerTestLamp-xxxx” where “xxxx” represents a random integer.
5. The test device calls GetLampName() on the bus object to get the name of the same connected lamp.
6. The test device checks to see if it has received a LampNameChanged signal from the DUT.
7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call to SetLampName() is executed properly.
- The method call to GetLampName() returns the same string used when setting the name.
- The test device receives a LampNameChanged signal after calling SetLampName().

## 3.6 LSF_Controller-v1-06: Lamp details

**Objective**

Verify that calling the GetLampDetails() methods will not result in a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device calls GetLampDetails() on the bus object.
5. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device gets the lamp ID of a lamp connected to the controller.
- The method call to GetLampDetails () does not result in a BusException.

# 3.7 LSF_Controller-v1-07: Lamp parameters

**Objective**

Verify that the DUT can call GetLampParameters() and GetLampParametesField() properly. The data returned from calling GetLampParameters() should be consistent with that returned by GetLampParametersField() for each specific field key.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device calls GetLampParameters() on the bus object.
5. The test device calls GetLampParametersField() on the bust object for every lamp parameter field.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call to GetLampParameters() is executed properly.
- Every method call to GetLampParametersField() returns the same value seen in the results of GetLampParameters().

# 3.8 LSF_Controller-v1-08: Lamp state

**Objective**

Verfiy that the DUT can call GetLampState() and GetLampStateField() properly. The data returned from calling GetLampState() should be consistent with that returned by GetLampStateField() for each specific field key.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device calls GetLampState() on the bus object.
5. The test device calls GetLampStateField() on the bust object for every lamp state field.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call to GetLampState() is executed properly
- Every method call to GetLampStateField() returns the same value seen in the results of GetLampState().

# 3.9 LSF_Controller-v1-09: Lamp state transition

**Objective**

Verify that the DUT can call TransitionLampState() and TransitionLampStateField() properly. Upon lamp transitions, the DUT should send the appropriate signal and GetLampStateField() should return the new state data.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device creates a new lamp state with the following:

```
OnOff = true
Brightness = 147483648L
Hue = 739688812
Saturation = 2061584302
ColorTemp = 384286547
```

5. The test device calls TransitionLampState() on the bust object using this new lamp state.
6. The test device waits to see if it has received a LampStateChanged signal from the DUT.
7. The test device calls TransitionLampStateField() to set OnOff to false.
8. The test device waits to see if it has received a LampStateChanged signal from the DUT, again.
9. The test device calls GetLampStateField() on the bus object for every lamp state field.
10. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method calls to TransitionLampState() and TransitionLampStateField() is executed properly
- The test device receives LampStateChanged signal after call to TransitionLampState() and TransitionLampStateField().
- Every method call to GetLampStateField() returns the same value seen in the results of GetLampState().

## 3.10 LSF_Controller-v1-10: Lamp state pulse

**Objective**

Verify that the DUT can call PulseLampWithState() without causing a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.

2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device creates two lamp states maps; one to represent the toState and another for the fromState.
5. The test device calls PulseLampWithState() on the bus object.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call PulseLampWithState() does not result in a BusException.

# 3.11 LSF_Controller-v1-11: Lamp change with preset

**Objective**

Verify that the DUT can call TransitionLampStateToPreset() and PulseLampWithPreset() without causing a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device creates two distinct presets, presetA and presetB, by calling CreatePreset() on the bus object.
5. The test device calls TransitionLampStateToPreset() using presetA on the bus object.
6. The test device calls PulseLampWithPreset() using presetA and presetB on the bus object.
7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call TransitionLampStateToPreset() does not result in a BusException.
- The method call PulseLampWithPreset() does not result in a BusException.

# 3.12 LSF_Controller-v1-12: Reset lamp state

**Objective**

Verify that the DUT can call ResetLampState() and ResetLampStateField() without causing a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device calls ResetLampState() on the bus object.
5. The test device transitions the lamp to an arbitrary state by calling TransitionLampState() on the bus object.
6. The test device calls ResetLampStateField() for Brightness on the bus object.
7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call ResetLampState() does not result in a BusException.
- The method call ResetLampStateField() does not result in a BusException.
- The method call ResetLampStateField() properly resets the Brightness property.

# 3.13 LSF_Controller-v1-13: Lamp faults

**Objective**

Verify that the DUT can call GetLampFaults() and ClearLampFault() without causing a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device calls GetLampFaults() on the bus object.
5. For evert lamp fault returned by GetLampFaults(), the test device calls ClearLampFault() on the bus object.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call GetLampFaults() does not result in a BusException.
- The method call ClearLampFault() does not result in a BusException.

## 3.14 LSF_Controller-v1-14: Lamp group CRUD

**Objective**

Verify that the DUT can perform create, read, update, and delete operations on lamp groups properly. Furthermore, ensure the DUT sends the appropriate signal for each operation.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller.
4. The test device calls CreateLampGroup() with the lamp ID from step 3.

5. The test device calls GetLampGroup() and checks to see if the returned group is that created in step 4.
6. The test device waits to see if it has received a LampGroupsCreated signal.
7. The test device calls UpdateLampGroup() on the bus object.
8. The test device waits to see if it has received a LampGroupsUpdated signal.
9. The test device calls DeleteLampGroup() on the bus object.
10. The test device calls GetAllLampGroups() to determine if the number of lamp groups has decreased by 1.
11. The test device waits to see if it has received a LampGroupsDeleted signal.
12. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller.
- The method call CreateLampGroup() creates a group on the DUT.
- The method call GetLampGroup() fetches the correct group; that is, the newly created group.
- The method call UpdateLampGroup() properly updates the group.
- The method call DeleteLampGroup() removes the group on the DUT.
- The method call GetAllLampGroups() returns the correct number of groups on the DUT.
- The signals LampGroupsCreated, LampGroupsUpdate, and LampGroupsDeleted are sent after CreateLampGroup(), UpdateLampGroup(), DeleteLampGroup() respectively.

# 3.15 LSF_Controller-v1-15: Get and set lamp group name

**Objective**

Verify that the DUT can call GetLampGroupName() and SetLampGroupName() properly. GetLampGroupName() should fetch the updated name after a call to SetLampGroupName(). Furthermore, verify that the appropriate signal is sent out after the lamp group name is changed.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.

2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller and creates a group with the lamp ID.
4. The test device calls SetLampGroupName() on the bus object..
5. The test device checks to see if it has received a LampGroupNameChanged signal from the DUT.
6. The test device calls GetLampGroupName() on the bus object.
7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller and successfully creates a group.
- The method call to SetLampGroupName() is executed properly.
- The method call to GetLampGroupName() returns the same string used when setting the name.
- The test device receives a LampGroupNameChanged signal after calling SetLampGroupName().

## 3.16 LSF_Controller-v1-16: Transition lamp group state

**Objective**

Verify that the DUT can call TransitionLampGroupState() and TransitionLampGroupStateField() to update the state of any lamps with a group.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller and creates a group with the lamp ID.
4. The test device creates a new lamp state with arbitrary values.
5. The test device calls TransitionLampGroupState() on the bus object.

6. The test device calls TransitionLampGroupStateField() on the bus object in order to change the OnOff field.
7. The test device verifies the lamp state has changed for a lamp within the group.
8. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller and successfully creates a group.
- The method call to TransitionLampGroupState() is executed properly.
- The method call to TransitionLampGroupStateField() is executed properly.
- The state of a lamp within the group reflects the new state.

# 3.17 LSF_Controller-v1-17: Pulse lamp group state

**Objective**

Verify calling PulseLampGroupWithState() will not result in a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller and creates a group with the lamp ID.
4. The test device creates two lamp states maps; one to represent the toState and another for the fromState.
5. The test device calls PulseLampGroupWithState() on the bus object.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.

- The test device obtains the lamp ID of the lamp connected to the controller and successfully creates a group.
- The method call to PulseLampGroupWithState() does not result in a BusException.

# 3.18 LSF_Controller-v1-18: Reset lamp group state

**Objective**

Verify that the DUT can call ResetLampGroupState() and ResetLampGroupStateField() without causing a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller and creates a group with the lamp ID.
4. The test device calls ResetLampGroupState() on the bus object.
5. The test device transitions the lamp to an arbitrary state by calling TransitionLampGroupState() on the bus object.
6. The test device calls ResetLampGroupStateField() for Brightness on the bus object.
7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller and successfully creates a group.
- The method call ResetLampGroupState() does not result in a BusException.
- The method call ResetLampGroupStateField() does not result in a BusException.
- The method call ResetLampGroupStateField() properly resets the Brightness property.

# 3.19 LSF_Controller-v1-19: Change lamp group state with presets

**Objective**

Verify that the DUT can call TransitionLampGroupStateToPreset() and PulseLampGroupWithPreset() without causing a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device gets the lamp ID of a lamp connected to the controller and creates a group with the lamp ID.
4. The test device creates two presets, presetA and presetB.
5. The test device calls TransitionLampGroupStateToPreset() on the bus object using presetA.
6. The test device calls PulseLampGroupWithPreset() on the bus object using presetA and presetB.
7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device obtains the lamp ID of the lamp connected to the controller and successfully creates a group.
- The method call TransitionLampGroupStateToPreset() does not result in a BusException.
- The method call PulseLampGroupWithPreset() does not result in a BusException.

# 3.20 LSF_Controller-v1-20: Default lamp state

**Objective**

Verify that the DUT can call GetDefaultLampState() and SetDefaultLampState() properly. GetDefaultLampState() should fetch the updated state after a call to SetDefaultLampState(). Furthermore, verify that the appropriate signal is sent out after the default lamp state is changed.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.

2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device calls SetDefaultLampState() on the bus object with an arbitrary lamp state.
4. The test device checks to see if it has received a DefaultLampStateChanged signal from the DUT.
5. The test device calls GetDefaultLampState() on the bus object.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The method call to SetDefaultLampState() is executed properly.
- The method call to GetDefaultLampState() returns the same string used when setting the name.
- The test device receives a DefaultLampStateChanged signal after calling SetDefaultLampState().

# 3.21 LSF_Controller-v1-21: Preset CRUD

**Objective**

Verify that the DUT can perform create, read, update, and delete operations on the preset interface properly. Furthermore, ensure the DUT sends the appropriate signal for each operation.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device calls CreatePreset() on the bus object using a arbitraty lamp state.
4. The test device waits to see if it has received a PresetsCreated signal.
5. The test device calls GetPreset() on the bus object.
6. The test device calls UpdatePreset() on the bus object.
7. The test device waits to see if it has received a PresetsUpdated signal.
8. The test device calls DeletePreset() on the bus object.

9. The test device waits to see if it has received a PresetsDeleted signal.
10. The test device calls GetAllPresetIDs() to determine if the number of presets is now 0.
11. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The method call CreatePreset() executes propely.
- The method call GetPreset() fetches the correct preset; that is, the newly created preset.
- The method call UpdatePreset() properly updates the preset.
- The method call DeletePreset() removes the preset on the DUT.
- The method call GetAllPresetIDs() returns the correct number of presets stored on the DUT.
- The signals PresetsCreated, PresetsUpdated, and PresetsDeleted are sent after CreatePreset(), UpdatePreset(), DeletePreset() respectively.

# 3.22 LSF_Controller-v1-22: Get and set preset name

**Objective**

Verify that the DUT can call GetPresetName() and SetPresetName() properly. GetPresetName() should fetch the updated name after a call to SetPresetName(). Furthermore, verify that the appropriate signal is sent out after the preset name is changed.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device creates a new preset.
4. The test device calls SetPresetName() on the bus object..
5. The test device checks to see if it has received a PresetsNameChanged signal from the DUT.
6. The test device calls GetPresetName() on the bus object.
7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device successfully creates a new preset.
- The method call to SetPresetName() is executed properly.
- The method call to GetPresetName() returns the same string used when setting the name.
- The test device receives a PresetNameChanged signal after calling SetPresetName().

# 3.23 LSF_Controller-v1-23: Create scene

**Objective**

Verify that the DUT can call CreateScene() without causing a BusException, and the correct signal is sent out when a scene is created on the DUT.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device calls CreateScene() on the bus object that consists of a transition effect.
4. The test device calls CreateScene() on the bus object that consists of a pulse effect.
5. The test device checks to see if it has received a ScenesCreated signal from the DUT.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The method call to CreateScene() is executed properly.
- The test device receives a ScenesCreated signal after calling CreateScene().

# 3.24 LSF_Controller-v1-24: Update and delete scene

**Objective**

Verify that the DUT can call UpdateScene() and DeleteScene() properly. Furthermore, ensure the DUT sends the appropriate signal for each operation.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device creates a scene consisting of a pulse effect.
4. The test device calls UpdateScene() on the bus object in order to update the fromState of the pulse effect.
5. The test device waits to see if it has received a ScenesUpdated signal.
6. The test device calls DeleteScene() on the bus object.
7. The test device waits to see if it has received a ScenesDeleted signal.
8. The test device calls GetAllSceneIDs() to determine if the number of scenes is now 0.
9. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device successfully creates a scene on the DUT.
- The method call UpdateScene() properly updates the scene.
- The method call DeletePreset() removes the preset on the DUT.
- The method call GetAllSceneIDs() returns the correct number of presets stored on the DUT.
- The signals ScenesUpdated and ScenesDeleted are sent after UpdateScene() and DeleteScene() respectively.

# 3.25 LSF_Controller-v1-25: Apply scene

**Objective**

Verify that the DUT can call ApplyScene() without causing a BusException and sends out the correct signal when the scene is applied.

### Procedure

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device creates a scene consisting of a pulse effect.
4. The test device calls ApplyScene() on the bus object.
5. The test device waits to see if it has received a ScenesApplied signal.
6. The test device leaves the session.

### Expected results

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device successfully creates a scene on the DUT.
- The method call ApplyScene() executes properly.
- The signal ScenesApplied is sent after the ApplyScene() method call.

## 3.26 LSF_Controller-v1-26: Get and set scene name

### Objective

Verify that the DUT can call GetSceneName() and SetSceneName() properly. GetSceneName() should fetch the updated name after a call to SetSceneName(). Furthermore, verify that the appropriate signal is sent out after the scene name is changed.

### Procedure

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device creates a scene consisting of a pulse effect.
4. The test device calls SetSceneName() on the bus object..
5. The test device checks to see if it has received a ScenesNameChanged signal from the DUT.
6. The test device calls GetSceneName() on the bus object.

7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device successfully creates a new scene.
- The method call to SetSceneName() is executed properly.
- The method call to GetSceneName() returns the same string used when setting the name.
- The test device receives a ScenesNameChanged signal after calling SetSceneName().

# 3.27 LSF_Controller-v1-27: Create master scene

**Objective**

Verify that the DUT can call CreateMasterScene() without causing a BusException, and the correct signal is sent out when a master scene is created on the DUT.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device calls CreateMasterScene() on the bus objet.
4. The test device checks to see if it has received a MasterScenesCreated signal from the DUT.
5. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The method call to MasterCreateScene() is executed properly.
- The test device receives a MasterScenesCreated signal after calling MasterCreateScene().

# 3.28 LSF_Controller-v1-28: Update and delete master scene

### Objective

Verify that the DUT can call UpdateMasterScene() and DeleteMasterScene() properly. Furthermore, ensure the DUT sends the appropriate signal for each operation.

### Procedure

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device creates a master scene.
4. The test device calls UpdateMasterScene() on the bus object in order to update the scenes within the master scene.
5. The test device waits to see if it has received a MasterScenesUpdated signal.
6. The test device calls DeleteMasterScene() on the bus object.
7. The test device waits to see if it has received a MasterScenesDeleted signal.
8. The test device calls GetAllMasterSceneIDs() to determine if the number of scenes is now 0.
9. The test device leaves the session.

### Expected results

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device successfully creates a master scene on the DUT.
- The method call UpdateMasterScene() properly updates the scene.
- The method call DeleteMasterPreset() removes the preset on the DUT.
- The method call GetAllMasterSceneIDs() returns the correct number of presets stored on the DUT.
- The signals ScenesMasterUpdated and ScenesMasterDeleted are sent after UpdateMasterScene() and DeleteMasterScene() respectively.

# 3.29 LSF_Controller-v1-29: Apply master scene

### Objective

Verify that the DUT can call ApplyMasterScene() without causing a BusException and sends out the correct signal when the master scene is applied.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device creates a master scene.
4. The test device calls ApplyMasterScene() on the bus object.
5. The test device waits to see if it has received a MasteScenesApplied signal.
6. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device successfully creates a master scene on the DUT.
- The method call ApplyMasterScene() executes properly.
- The signal MasterScenesApplied is sent after the ApplyMasterScene() method call.

## 3.30 LSF_Controller-v1-30: Get and set master scene name

**Objective**

Verify that the DUT can call GetMasterSceneName() and SetMasterSceneName() properly. GetMasterSceneName() should fetch the updated name after a call to SetMasterSceneName(). Furthermore, verify that the appropriate signal is sent out after the master scene name is changed.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device creates a master scene.
4. The test device calls SetMasterSceneName() on the bus object..

5. The test device checks to see if it has received a MasterScenesNameChanged signal from the DUT.
6. The test device calls GetMasterSceneName() on the bus object.
7. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device successfully creates a new master scene.
- The method call to SetMasterSceneName() is executed properly.
- The method call to GetMasterSceneName() returns the same string used when setting the name.
- The test device receives a MasterScenesNameChanged signal after calling SetMasterSceneName().

## 3.31 LSF_Controller-v1-31: Leader election get checksum and modification timestamp

**Objective**

Verfiy that the DUT can call GetChecksumAndModificationTimestamp() and GetBlob() properly. The checksum returned from calling GetBlob() should be consistent with that returned by GetChecksumAndModificationTimestamp() for each specific blob type.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device calls GetChecksumAndModificationTimestamp() on the bus object.
4. The test device calls GetBlob() on the bust object for every blob type.
5. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.

- The method call to GetChecksumAndModificationTimestamp() is executed properly.
- The method call to GetBlob() is executed properly.
- Every method call to GetBlob() returns the same timestamp seen in the results of GetChecksumAndModificationTimestamp().

# 3.32 LSF_Controller-v1-32: Leader election blob changed

**Objective**

Verfiy that the DUT sends out a BlobChanged signal when it's internal data has changed.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.
3. The test device creates a new scene in order to cause the blob to change.
4. The test waits to see if it has received a BlobChanged signal from the DUT.
5. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The test device receives a BlobChanged signal after a new scene is created.

# 3.33 LSF_Controller-v1-33: Leader election overthrow

**Objective**

Verfiy that the DUT can call method Overthrow() without resulting in a BusException.

**Procedure**

1. The test device listens for an About announcement from the application on the DUT.
2. After receiving an About announcement from the application, the test device joins a session with the application at the port specified in the received About announcement.

3. The test calls Overthrow() on the bus object.
4. The test device leaves the session.

**Expected results**

- The test device receives an About announcement from the application on the DUT.
- The test device joins a session with the application at the port specified in the received About announcement.
- The method call to Overthrow() does not cause a BusException.