BACnet® TESTING LABORATORIES
ADDENDA

Addendum m to BTL Test Package 15.2

Revision 2
Revised 06/29/2019

Approved by the BTL Working Group on June 20, 2019>;
Approved by the BTL Working Group Voting Members on <date>;
Published on <date>.

[This foreword and the "Overview" on the following pages are not part of this Test Package. They are merely informative and do not contain requirements necessary for conformance to the Test Package.]

FOREWORD

The purpose of this addendum is to present current changes being made to the BTL Test Package. These modifications are the result of change proposals made pursuant to the continuous maintenance procedures and of deliberations within the BTL-WG Committee. The changes are summarized below.

In the following document, language to be added to existing clauses within the BTL Test Package 15.2 is indicated through the use of *italics*, while deletions are indicated by ~~strikethrough~~. Where entirely new subclauses are proposed to be added, plain type is used throughout

In addition, changes to BTL Specified Tests also contain a yellow highlight to indicate the changes made by this addendum.

When this addendum is applied, all highlighting will be removed. Change markings on tests will remain to indicate the difference between the new test and an existing 135.1 test. If a test being modified has never existed in 135.1, the applied result should not contain any change markings. When this is the case, square brackets will be used to describe the changes required for this test.

Each addendum can stand independently unless specifically noted via dependency within the addendum. If multiple addenda change the same test or section, each future released addendum that changes the same test or section will note in square brackets whether or not those changes are reflected.

BTL-15.2m-1: Add Access Object Tests [BTLWG-438]

Overview:

To create new tests for the following 6 access objects added in **protocol rev 9**:

- **Access Point**
- **Access Zone**
- **Access User**
- **Access Rights**
- **Access Credential**
- **Credential Data Input**

Changes:

[In BTL Checklist, modify existing Access Point object type to Section 3, Objects]

Support	Listing	Option
Access Point Object		
	R~~[1]~~	Base Requirements
	S	*Supports writable Out_Of_Service properties*
	O	*Supports writable Active_Authentication_Policy Property*
	O	*Supports writable Lockout Property*
	O	*Supports writable Threat_Level Property*
	O	*Supports Occupancy_Upper_Limit_Enforced and Occupancy_Count_Adjust Properties*
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
	~~[1]Contact BTL for interim tests for this object.~~	

[In BTL Test Plan, modify existing Access Point object section 3.44]

3.44 Access Point Object

3.44.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *Base requirements must be met by any IUT that can contain Access Point objects*

BTL - 7.3.2.X56.1 - Authentication_Status and Access_Event Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X56.2 - Allowed Access Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X56.3 - Denied Access Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X56.4 - Authorization Mode Test		
	Test Conditionality	*If the IUT only supports the authorization modes AUTHORIZE, this test shall be skipped.*
	Test Directives	
	Testing Hints	

BTL - 7.3.2.X56.5 - Access Rights Exemptions Test		
	Test Conditionality	*If the Authorization_Exemption property is not supported or the ACCESS_RIGHTS exemption is not supported this test shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X56.10 Denied Access Disabled Credential Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.44.2 Supports Writable Out_Of_Service Properties

The Out_Of_Service property in Access Point objects contained in the IUT are writable.

135.1-2013 - 7.3.1.1 - Out_Of_Service, Status_Flags, and Reliability Tests		
	Test Conditionality	*Must be executed.*
	Test Directives	*The test shall be executed using an Access Point object*
	Testing Hints	

3.44.3Supports Writable Active_Authentication_Policy Property

IUT supports Access Point objects with writable Active_Authentication_Policy property.

BTL - 7.3.2.X56.6 - Change Authentication Policy Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.44.4Supports Writable Lockout Property

IUT supports Access Point objects with writable Lockout Property.

BTL - 7.3.2.X56.7 - Lockout State Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.44.5Supports Writable Threat_Level

IUT supports Access Point objects with writable Threat_Level

BTL - 7.3.2.X56.8 - Threat Level Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.44.6Supports Occupancy_Upper_Limit_Enforced and Occupancy_Count_Adjust

IUT supports Access Point objects with modifiable Occupancy_Upper_Limit_Enforced and Occupancy_Count_Adjust

BTL - 7.3.2.X56.9 - Denied Access Occupancy Upper Limit Test		
	Test Conditionality	*If the IUT does not support the Access Zone object type, this test shall be skipped.*
	Test Directives	
	Testing Hints	

3.44.7 Contains an object with Reliability_Evaluation_Inhibit Property

The IUT contains, or can be made to contain, a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	

[In BTL Specified Tests, add Access Point object specific tests in section 7.3.2.X56]

7.3.2.X56 Access Point Object Tests

The Access Point object type represents the external interface of the access control decision engine for a specific door. A credential is entered, the access rights of the credential are determined and the access decision is determined based on the access rights. Testing this authentication and authorization functionality requires the support of other standard BACnet access control object types. The required and optional object types are shown in figure X1.

The access decision begins with a credential value being sent to the Access Point object for evaluation. Typically the credential value is read at a Credential Data Input object which extracts the raw credential data from the physical reader, formats the data and then sends it to the corresponding Access Point object. If the Credential Data Input object type is not supported then the vendor must provide an alternate method for the credential to be received by the Access Point.

When a credential value is received by the Access Point object it searches through the Access Credential objects to find the one with a matching credential value. For each credential value being tested a corresponding Access Credential must exist within the IUT. The only exception to this is when testing for an unknown credential (DENIED_UNKNOWN_CREDENTIAL).

To determine if access is granted or denied for a specific credential each Access Credential object must reference an Access Rights object which defines the appropriate access rights corresponding to the specific test being executed.

Some of the allowed and denied access tests require the Access Zone object. In this case an Access Point must be configured to be an entry access point to the access zone. These tests require that the Access Rights objects reference the Access Zone rather than the Access Point.

When the access decision is determined the IUT shall provide a method to indicate the result. Typically the decision is exposed through the Access Door object. When access is granted the door is pulsed unlocked and when denied the door remains locked. If the Access Door object is not used then another method of showing the result shall be configured.

Unless specified otherwise in the specific test, the access point object (AP1) in following tests shall have the following configuration:

a) Authorization_Mode shall have the value AUTHORIZE.

b) Out_Of_Service shall be FALSE.

c) Lockout shall be FALSE.

d) Muster_Point shall be FALSE.

e) Authentication_Status shall be READY.

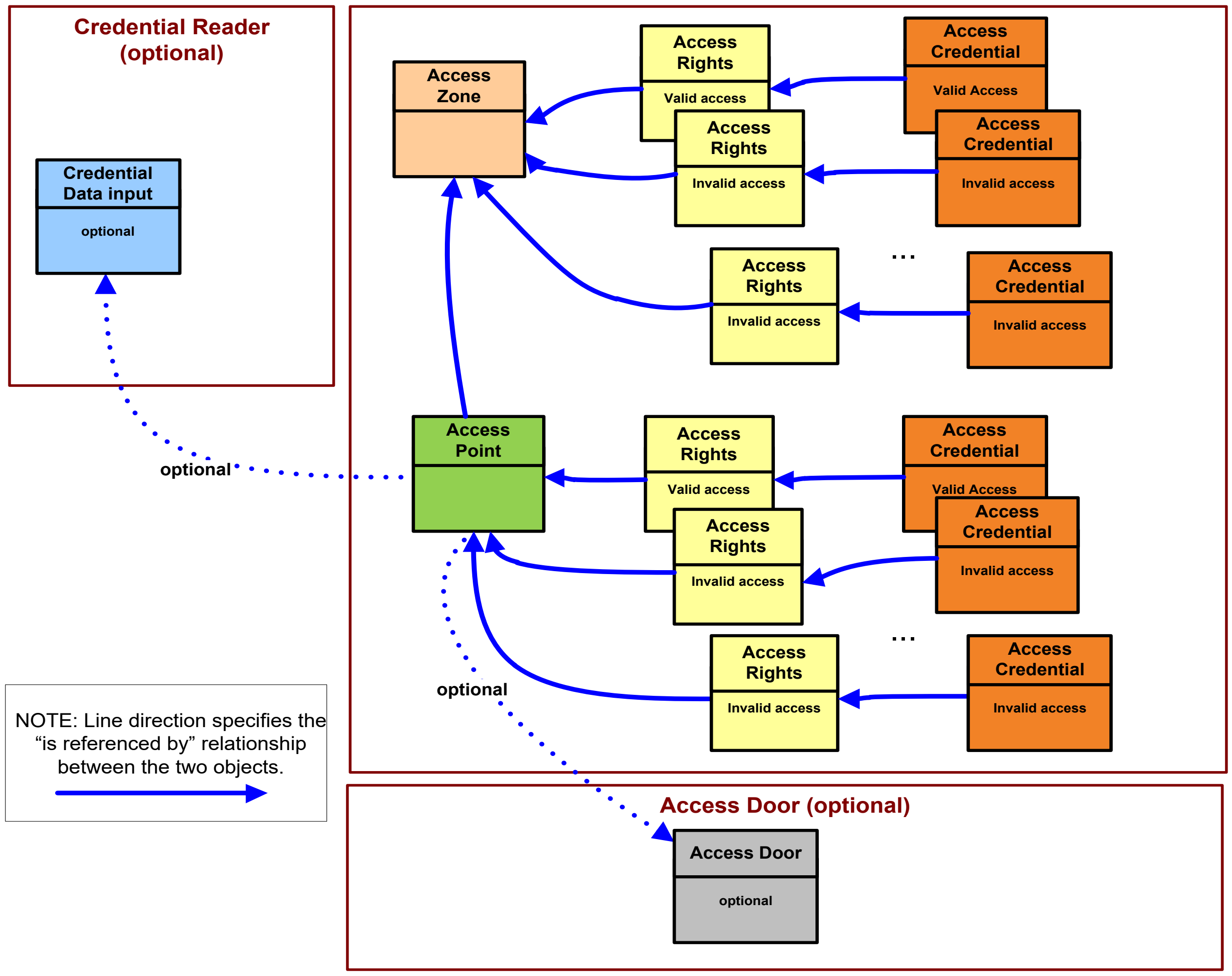

Figure X1: Objects and relationships used for testing Access Point objects

7.3.2.X56.1 Authentication_Status and Access_Event Test

Purpose: This test case verifies the authentication and authorization process are disabled when Out_Of_Service of the Access Point object is TRUE. It also verifies the interrelationship between the Out_Of_Service, Authentication_Status,Access_Event and Access_Event_Time properties.

Test Concept: Write TRUE to the Out_Of_Service property and verify that the authorization and authentication functions are disabled. Write FALSE to the Out_Of_Service property and verity that they are enabled.

Configuration Requirements:
See 7.3.2.X56

Test Steps:

```
1. VERIFY Authentication_Status = READY
2. IF (Out_Of_Service is writable) THEN
       WRITE Out_Of_Service = TRUE
   ELSE
       MAKE (Out_Of_Service TRUE)
3. VERIFY Authentication_Status = DISABLED
4. VERIFY Access_Event = OUT_OF_SERVICE
5. VERIFY Access_Event_Time = (the time Out_Of_Service was set to TRUE)
6. IF (Out_Of_Service is writable) THEN
       WRITE Out_Of_Service = FALSE
   ELSE
       MAKE (Out_Of_Service FALSE)
7. VERIFY Authentication_Status = READY
8. VERIFY Access_Event = OUT_OF_SERVICE_RELINQUISHED
9. VERIFY Access_Event_Time = (the time Out_Of_Service was set to FALSE)
```

7.3.2.X56.2 Allowed Access Test

Purpose: To verify that a valid credential that is allowed access to this access point at this time is granted access.

Test Concept: A valid credential, that has access to the access point being tested at the current time, is presented at the access point. It is then verified that access is allowed and the appropriate access event is generated.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.

Test Steps:

```
1. READ EventTag = Access_Event_Tag
2. MAKE (present a valid credential at credential reader for this access point)
3. VERIFY Access_Event = GRANTED
4. VERIFY Access_Event_Time = (the time that the credential was presented)
5. VERIFY Access_Event_Credential = C1
6. VERIFY Access_Event_Tag = EventTag + 1
```

7.3.2.X56.3 Denied Access Test

Purpose: To verify that a credential that is not allowed access to this access point at this time is denied access. There are a number of reasons why a credential may be denied access and this test tests the situations which must be supported by the access point.

Test Concept: To test that a credential, which is not allowed access to this access point, is presented at the access point with the result that access is allowed and the appropriate access event is generated.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) The vendor shall provide a set of credentials which correspond to Access Credential objects configured such that access to the access point shall denied for the following reasons:
    a. DENIED_POINT_NO_ACCESS_RIGHTS = C1
    b. DENIED_NO_ACCESS_RIGHTS = C2
    c. DENIED_ZONE_NO_ACCESS_RIGHTS = C3
    d. DENIED_CREDENTIAL_NOT_YET_ACTIVE = C4
    e. DENIED_CREDENTIAL_EXPIRED = C5
    f. DENIED_CREDENTIAL_MANUAL_DISABLE = C6
    g. DENIED_INCORRECT_AUTHENTICATION_FACTOR = C7
    h. DENIED_OUT_OF_TIME_RANGE = C8
    i. DENIED_THREAT_LEVEL = C9
    j. DENIED_PASSBACK = C10

k. DENIED_UNEXPECTED_LOCATION_USAGE = C11
l. DENIED_MAX_ATTEMPTS = C12
m. DENIED_AUTHENTICATION_FACTOR_LOST = C13
n. DENIED_AUTHENTICATION_FACTOR_STOLEN= C14
o. DENIED_AUTHENTICATION_FACTOR_DAMAGED = C15
p. DENIED_AUTHENTICATION_FACTOR_DESTROYED = C16
q. DENIED_AUTHENTICATION_FACTOR_DISABLED = C17
r. DENIED_AUTHENTICATION_FACTOR_ERROR = C18
s. DENIED_CREDENTIAL_UNASSIGNED = C19
t. DENIED_CREDENTIAL_NOT_PROVISONED = C20
u. DENIED_CREDENTIAL_LOCKOUT = C21
v. DENIED_CREDENTIAL_MAX_DAYS = C22
w. DENIED_CREDENTIAL_MAX_USES = C23
x. DENIED_CREDENTIAL_DISABLED = C24
y. DENIED_LOCKOUT = C25

Note to Tester: if the IUT does not support any of the above denial reasons then the corresponding credentials are not required to be supplied.

Test Steps:

```
1.  REPEAT C = (C1…C25) DO {
        READ EventTag = Access_Event_Tag
        MAKE (present the credential corresponding to C at the credential reader for this access point)
        VERIFY Access_Event = (denied reason corresponding to credential C)
        VERIFY Access_Event_Time = (the time that credential C was presented)
        VERIFY Access_Event_Credential = C
        VERIFY Access_Event_Tag = EventTag + 1
        }

-- verify unknown credential event
2.  READ EventTag = Access_Event_Tag
3.  MAKE (present a credential which does not correspond to any configured Access Credential object at the credential reader for this access point)
4.  VERIFY Access_Event = DENIED_UNKNOWN_CREDENTIAL
5.  VERIFY Access_Event_Time = (the time that the credential was presented)
6.  VERIFY (Instance Number within the Access_Event_Credential object identifier and device identifier= 4194303)
7.  VERIFY Access_Event_Tag = EventTag + 1
```

7.3.2.X56.4 Authorization Mode Test

Purpose: To verify each authorization mode supported by this IUT.

Test Concept:
For each authorization mode supported by the IUT a valid credential is presented at the access point to verify that the appropriate action is taken.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.
b) An active credential with no valid access rights shall be represented by Access Credential object C2.

Note: If the VERIFICATION_REQUIRED or AUTHORIZATION_DELAYED mode is supported the vendor must provide a mechanism for external verification to be performed.

Test Steps:

-- verify GRANT_ACTIVE mode

```
1.  IF (GRANT_ACTIVE is supported) THEN
        READ EventTag = Access_Event_Tag
        WRITE Authorization_Mode = GRANT_ACTIVE
        MAKE (present credential C2 at credential reader for this access point)
        VERIFY Access_Event = GRANTED
        VERIFY Access_Event_Time = (the time that credential C2 was presented)
        VERIFY Access_Event_Credential = C2
        VERIFY Access_Event_Tag = EventTag + 1

-- verify DENY_ALL mode
2.  IF (DENY_ALL is supported) THEN
        READ EventTag = Access_Event_Tag
        WRITE Authorization_Mode = DENY_ALL
        MAKE (present credential C1 at credential reader for this access point)
        VERIFY Access_Event = DENIED_DENY_ALL
        VERIFY Access_Event_Time = (the time that credential C1 was presented)
        VERIFY Access_Event_Credential = C1
        VERIFY Access_Event_Tag = EventTag + 1

-- verify VERIFICATION_REQUIRED mode (verification authorized)
3.  IF (VERIFICATION_REQUIRED is supported) THEN
        READ EventTag = Access_Event_Tag
        WRITE Authorization_Mode = VERIFICATION_REQUIRED
        MAKE (present credential C1 at credential reader for this access point)
        VERIFY Access_Event = VERIFICATION_REQUIRED
        VERIFY Access_Event_Time = (the time that credential C1 was presented most recently)
        VERIFY Access_Event_Credential = C1
        VERIFY Authentication_Status = WAITING_FOR_VERIFICATION
        MAKE (external verification process grants access)
        VERIFY Access_Event = GRANTED
        VERIFY Access_Event_Time = (the time that verification process granted access)
        VERIFY Access_Event_Credential = C1
        VERIFY Access_Event_Tag = EventTag + 1

-- verify VERIFICATION_REQUIRED mode (verification denied)
        READ EventTag = Access_Event_Tag
        WRITE Authorization_Mode = VERIFICATION_REQUIRED
        MAKE (present credential C1 at credential reader for this access point)
        VERIFY Access_Event = VERIFICATION_REQUIRED
        VERIFY Access_Event_Time = (the time that credential C1 was presented)
        VERIFY Access_Event_Credential = C1
        VERIFY Authentication_Status = WAITING_FOR_VERIFICATION
        MAKE (external verification process denies access)
        VERIFY Access_Event = DENIED_VERIFICATION_FAILED
        VERIFY Access_Event_Time = (the time that verification process denied access)
        VERIFY Access_Event_Credential = C1
        VERIFY Access_Event_Tag + 1

-- verify VERIFICATION_REQUIRED mode (verification timeout)
        READ EventTag = Access_Event_Tag
        WRITE Authorization_Mode = VERIFICATION_REQUIRED
        MAKE (present credential C1 at credential reader for this access point)
        VERIFY Access_Event = VERIFICATION_REQUIRED
        VERIFY Access_Event_Time = (the time that credential C1 was presented)
        VERIFY Access_Event_Credential = C1
        VERIFY Authentication_Status = WAITING_FOR_VERIFICATION
        MAKE (external verification process does not respond within verification time)
        WAIT Verification_Time
        VERIFY Access_Event = DENIED_VERIFICATION_TIMEOUT
        VERIFY Access_Event_Time = (the time that verification process timed out)
        VERIFY Access_Event_Credential = C1
        VERIFY Access_Event_Tag = EventTag + 1

-- verify AUTHORIZATION_DELAYED mode (access granted)
4.  IF (AUTHORIZATION_DELAYED is supported) THEN
        WRITE Authorization_Mode = AUTHORIZATION_DELAYED
        MAKE (present credential C1 at credential reader for this access point)
        VERIFY Access_Event = AUTHORIZATION_DELAYED
        VERIFY Access_Event_Time = (the time that credential C1 was presented)
        VERIFY Access_Event_Credential = C1
        MAKE (external verification process does not respond within verification time)
        WAIT Verification_Time
        VERIFY Access_Event = GRANTED
        VERIFY Access_Event_Time = (the time that verification process timed out)
        VERIFY Access_Event_Credential = C1

-- verify AUTHORIZATION_DELAYED mode (access denied)
        WRITE Authorization_Mode = AUTHORIZATION_DELAYED
        READ EventTag = Access_Event_Tag
        MAKE (present credential C1 at credential reader for this access point)
        VERIFY Access_Event = AUTHORIZATION_DELAYED
        VERIFY Access_Event_Time = (the time that credential C1 was presented)
        VERIFY Access_Event_Credential = C1
        MAKE (external verification process denies access)
        VERIFY Access_Event = DENIED_VERIFICATION_FAILED
        VERIFY Access_Event_Time = (the time that verification process denied access)
        VERIFY Access_Event_Credential = C1
        VERIFY Access_Event_Tag = EventTag + 1

-- verify NONE mode
5.  IF (NONE is supported) THEN
        WRITE Authorization_Mode = NONE
        WAIT Internal Processing Fail Time
        VERIFY Authentication_Status = DISABLED
```

7.3.2.X56.5 Access Rights Exemptions Test

Purpose: To verify the access rights exemption functionality.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) An active credential with no access rights shall be represented by Access Credential object C1.
b) The Authorization_Exemption list of C1 shall be empty.

Test Steps:

```
-- verify access is denied for the credential
1.  MAKE (present credential C1 at credential reader for access point AP1)
2.  VERIFY Access_Event = DENIED_NO_ACCESS_RIGHTS
3.  VERIFY Access_Event_Time = (the time that the credential was presented)
4.  VERIFY Access_Event_Credential = C1

-- verify access is granted for the credential when the master exemption set to TRUE
5.  MAKE C1, Authorization_Exemption = (ACCESS_RIGHTS)
6.  MAKE (present credential C1 at credential reader for access point AP1)
7.  VERIFY Access_Event = GRANTED
8.  VERIFY Access_Event_Time = (the time that the credential was presented)
9.  VERIFY Access_Event_Credential = C1
```

7.3.2.X56.6 Change Authentication Policy Test

Purpose: To verify that the Active_Authentication_Policy property of the Access Point object accepts valid authentication policy values and does not accept invalid ones. It also verifies that an error response is returned if the authentication policy is changed to a non-existent policy number.

Test Concept: The Active_Authentication_Policy is written with values of 1 to X, where X is the number of valid authentication policies to verify that the value is accepted. Then it is written with a value larger than X to verify that the value is rejected with a VALUE_OUT_OF_RANGE error. Finally, it is written with a value of 0 to verify that the value is also rejected with the same error.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) The IUT shall be configured with at least one active authentication policy.
b) All authentication policies shall be valid policies.

Test Steps:

```
-- verify that the active authentication policy can set to all valid policies
1. READ Count = Number_Of_Authentication_Policies
2. REPEAT X = (1 to Count) DO {
        WRITE Active_Authentication_Policy = X
        VERIFY Active_Authentication_ Policy = X
        }

-- verify that writing an invalid authentication policy to active authentication policy results in a reject
3. TRANSMIT WriteProperty-Request,
        'Object Identifier' =          Access_Point object,
        'Property Identifier' =        Active_Authentication_Policy,
        'Property Value' =             (any value larger than Count)
4  RECEIVE BACnet-Error-PDU,
        Error Class =                  PROPERTY,
        Error Code =                   VALUE_OUT_OF_RANGE

-- verify that writing 0 to the authentication policy results in a reject
5  TRANSMIT WriteProperty-Request,
        'Object Identifier' =          Access_Point object,
        'Property Identifier' =        Active_Authentication_Policy,
        'Property Value' =             0
6  RECEIVE BACnet-Error-PDU,
        Error Class =                  PROPERTY,
        Error Code =                   VALUE_OUT_OF_RANGE
```

7.3.2.X56.7 Lockout State Test

Purpose: To verify that access is denied for any credential when the access point is in the lockout state. To verify that using an invalid credential at the an access point multiple times will cause the access point to go into a lockout state. To verify that the lockout will automatically relinquish after the specified time.

Test Concept: A credential which will result in denied access is repeatedly presented at the access point until the access point becomes locked out. When the access point becomes locked valid credentials will also be denied access until the lockout relinquish time has expired.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) The Max_Failed_Attempts property, if present, has a value greater than 0
b) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.
c) An active credential with no valid access rights for the access point shall be represented by Access Credential object C2.
d) The Failed_Attempts_Events list, if present, shall have at least one entry corresponding to the reason why C2 is denied access.
e) The Lockout_Relinquish_Time has a value greater than 0

Test Steps:
-- verify that valid credentials are denied when the Lockout property is TRUE
1. WRITE Lockout = TRUE
2. WAIT **Internal Processing Fail Time**
3. VERIFY Access_Event = LOCKOUT_OTHER
4. VERIFY Access_Event_Time = (the time that TRUE was written to the Lockout property)
5. VERIFY Access_Event_Credential = (a BACnetDeviceObjectReference containing (Device, Instance number 4194303))
6. MAKE (present credential C1 at credential reader for this access point)
7. VERIFY Access_Event = DENIED_LOCKOUT
8. VERIFY Access_Event_Time = (the time that credential C1 was presented)
9. VERIFY Access_Event_Credential = C1

-- verify that using an invalid credential at the an access point multiple times will cause the access point to go into a lockout state

```
10. WRITE Lockout = FALSE
11. WAIT Internal Processing Fail Time
12. VERIFY Access_Event = LOCKOUT_RELINQUISH
13. VERIFY Access_Event_Time = (the time that FALSE was written to the Lockout property)
14. VERIFY Access_Event_Credential = (a BACnetDeviceObjectReference containing (Device, Instance number 4194303))
15. IF (Failed_Attempts and Max_Failed_Attempts are supported)
            REPEAT X= (1 to Max_Failed_Attempts + 1) DO {
            READ FailedAttempts = Failed_Attempts
            MAKE (present credential C2 at credential reader for this access point)
            VERIFY (Failed_Attempts = FailedAttempts + 1)
            }
16. VERIFY (Lockout = TRUE)
17. VERIFY (Access_Event = LOCKOUT_MAX_ATTEMPTS)
18. VERIFY (Access_Event_Time = the time that Lockout was set to TRUE)
19. VERIFY (Access_Event_Credential = C2)
20. MAKE (present credential C1 at credential reader for this access point)
21. VERIFY (Access_Event = DENIED_LOCKOUT)
22. VERIFY (Access_Event_Time = the time that credential C1 was presented)
23. VERIFY (Access_Event_Credential = C1)
```

-- verify that the lockout will automatically relinquish after the specified time

24. WAIT Lockout_Relinquish_Time
25. VERIFY (Lockout = FALSE)
26. VERIFY (Access_Event = LOCKOUT_RELINQUISHED)
27. VERIFY (Access_Event_Time = the time that Lockout was set to FALSE)
28. VERIFY Access_Event_Credential = (a BACnetDeviceObjectReference containing (Device, Instance number 4194303))
29. MAKE (present credential C1 at credential reader for this access point)
30. VERIFY (Access_Event = GRANTED)
31. VERIFY (Access_Event_Time = the time that credential C1 was presented)
32. VERIFY (Access_Event_Credential = C1)

7.3.2.X56.8 Threat Level Test
Purpose: To verify Threat_Level is used to Grant or Deny access based on the Threat_Authority of the Access Credential

Test Concept: Vary the Threat_Level of the access point to be lower or equal than the Threat_Authority of the credential to verify that access is granted at this access point. Change the Threat_Level of the access point to the higher than the Threat_Authority of the credential to verify that access is denied.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) An active credential with valid access rights for the access point shall be represented by Access Credential object C1.This credential shall have a Threat_Authority of X, where 0 < X <100.

Test Steps:
-- verify that a credential with threat authority greater than threat level of the access point is granted access

1. WRITE Threat_Level = (any value less than X)
2. MAKE (present credential C1 at credential reader for this access point)
3. VERIFY Access_Event = GRANTED
4. VERIFY Access_Event_Time = (the time that credential C1 was presented)
5. VERIFY Access_Event_Credential = C1

-- verify that a credential with threat authority equal to the threat level of the access point is granted access

6. WRITE Threat_Level = X
7. MAKE (present credential C1 at credential reader for this access point)
8. VERIFY Access_Event = GRANTED
9. VERIFY Access_Event_Time = (the time that credential C1 was presented)
10. VERIFY Access_Event_Credential = C1

-- verify that a credential with threat authority less than the threat level of the access point is denied access

11. WRITE Threat_Level = (any value greater than X)
12. MAKE (present credential C1 at credential reader for this access point)
13. VERIFY Access_Event = DENIED_THREAT_LEVEL
14. VERIFY Access_Event_Time = (the time that credential C1 was presented)
15. VERIFY Access_Event_Credential = C1

7.3.2.X56.9 Denied Access Occupancy Upper Limit Test
Purpose: To verify occupancy counting, occupancy restrictions and occupancy exemptions.

Test Concept: When occupancy counting is enabled and a valid credential is presented at the access point then this test verifies that access is granted only if the occupancy limits are not violated. If the occupancy limits are violated then access is denied. If the credential has an occupancy exemption then the credential will be granted access regardless of the occupancy count of the zone.

Configuration Requirements:
See 7.3.2.X56. This test requires the following additional configuration:

a) The Access Point object is configured to be an entry access point to an access zone which is represented by Access Zone object Z1.
b) The Occupancy_Upper_Limit property of Z1 shall have the value X, where X>0.
c) A number of active credentials all with valid access rights for the access point shall be represented by Access Credential objects C1...C(X+1) which shall be configured such that each has valid access to the access point.
d) The Occupancy_Upper_Limit_Enforced property shall have a value of TRUE.
e) The Occupancy_Count_Adjust property shall have a value of TRUE.
f) The Occupancy_Count property of Z1 shall have the value 0.
g) The Occupancy_Count_Enable property of Z1 shall have a value of TRUE.
h) The Authorization_Exemptions property, if it exists, of C(X+1) shall be empty.

Test Steps:
-- verify that a credential with valid access is granted access while the occupancy count of the zone is less than the Occupancy_Upper_Limit property in the zone

```
1.  REPEAT C = (C1…CX) DO {
    READ Count = Z1, Occupancy_Count
    MAKE (present credential C at credential reader for this access point)
    VERIFY Z1,Occupancy_Count  = (Count +1)
    VERIFY Access_Event = GRANTED
    VERIFY Access_Event_Time = (the time that credential C was presented)
    VERIFY Access_Event_Credential = C
    }
```

-- verify that a credential with valid access is denied access when the occupancy count of the zone becomes greater than the Occupancy upper limit

2. READ Count = Z1, Occupancy_Count
3. VERIFY Z1,Occupancy_Upper_Limit = Count
4. MAKE (present credential C(X+1) at credential reader for this access point)
5. VERIFY Access_Event = DENIED_UPPER_OCCUPANCY_LIMIT

6. VERIFY Access_Event_Time = (the time that credential C(X+1) was presented)
7. VERIFY Access_Event_Credential = C(X+1)
8. VERIFY Z1,Occupancy_Count = Count

-- verify that a credential with valid access and an occupancy exemption is granted access when the occupancy count of the zone becomes greater than the Occupancy upper limit

9. IF (If the Authorization_Exemption property is not supported or the OCCUPANCY_CHECK exemption is not supported)
10. READ Count = Z1, Occupancy_Count
11. VERIFY Z1,Occupancy_Upper_Limit = Count
12. WRITE C(X+1), Authorization_Exemptions = (OCCUPANCY_CHECK)
13. MAKE (present credential C(X+1) at credential reader for this access point)
14. VERIFY Access_Event = GRANTED
15. VERIFY Access_Event_Time = (the time that credential C(X+1) was presented)
16. VERIFY Access_Event_Credential = C(X+1)
17. VERIFY Z1,Occupancy_Count = Count + 1

7.3.2.X56.10 Denied Access Disabled Credential Test

Purpose: To test that a credential is denied access at an access point when the credential is disabled even when it has valid access rights.

Test Concept: A disabled credential is presented at an access point and access is denied. For testing purposes, the credential shall be disabled by setting the Expiration_Time to a time in the past.

Configuration Requirements:
See 7.3.2.X56.1. This test requires the following additional configuration:

a) An access credential with valid access to AP1 shall be represented by Access Credential C1.
b) The Credential_Status property shall have the value ACTIVE.
c) The Reason_For_Disable property shall be empty.

Test Steps:

– test granted access when credential is active

1. VERIFY Reason_For_Disable = ( )
2. VERIFY Credential_Status = ACTIVE
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY AP1,Access_Event = GRANTED
5. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
6. VERIFY AP1,Access_Event_Credential = C1

– test denied access when credential is inactive

7. MAKE (Expiration_Time = time < current time)
8. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
9. VERIFY Credential_Status = INACTIVE
10. MAKE (present credential C1 at credential reader for access point AP1)
11. VERIFY AP1,Access_Event = DENIED_CREDENTIAL_EXPIRED
12. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
13. VERIFY AP1,Access_Event_Credential = C1

[In BTL Checklist, modify existing Access Zone object type to Section 3, Objects]

Support	Listing	Option
Access Zone Object		
	R~~+~~	Base Requirements
	S	*Supports Writable Out_Of_Service Properties*
	O	*Supports Occupancy_Count*
	O	*Supports Credentials_In_Zone*
	O	*Supports Passback_Mode*
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
	~~+Contact BTL for interim tests for this object.~~	

[In BTL Test Plan, modify existing Access Zone object tests in section 3.45]

3.45 Access Zone Object

3.45.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *There are no base requirements tests for this section.*

3.45.2 Supports Writable Out_Of_Service Properties

The Out_Of_Service property in Access Zone objects contained in the IUT are writable.

135.1-2013 - 7.3.1.1 - Out_Of_Service, Status_Flags, and Reliability Tests		
	Test Conditionality	*Must be executed.*
	Test Directives	*The test shall be executed using an Access Zone object*
	Testing Hints	

3.45.3 Supports Occupancy_Count

The IUT supports Occupancy_Count property

BTL - 7.3.2.X57.1 - Occupancy State Test		
	Test Conditionality	*If the Occupancy_Lower_Limit is not supported then the steps of the test which test this functionality shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X57.2 - Occupancy Counting Test		
	Test Conditionality	*If the Occupancy_Lower_Limit is not supported then the steps of the test which test this functionality shall be skipped.*
	Test Directives	
	Testing Hints	

3.45.4 Supports Credentials_In_Zone

The IUT supports Credentials_In_Zone property.

BTL - 7.3.2.X57.3 - Keeping Track of Credentials Test		
	Test Conditionality	
	Test Directives	
	Testing Hints	

3.45.5 Supports Passback_Mode

The IUT supports Passback_Mode property.

BTL - 7.3.2.X57.4 - Passback Mode Test

	Test Conditionality	*If the IUT does not support soft passback or hard passback mode then those tests shall be skipped. If Passback_Timeout is not supported then those steps shall be skipped. If the passback exemption of the Access Credential object type is not supported then those steps shall be skipped.*
	Test Directives	
	Testing Hints	

3.45.6 Contains an object with Reliability_Evaluation_Inhibit Property

The IUT contains, or can be made to contain, a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	

[In BTL Specified Tests, add Access Zone object specific tests in section 7.3.2.57.X]

7.3.2.57 Access Zone Object Tests

Many of the tests for the Access Zone object require interactions from other BACnet access control objects as a result of a valid credential being processed. The required and optional BACnet object types are shown in figure X2.

The Access Zone is defined by list of Access Points that act as entry and exit points of the zone. The configuration requires at least one Access Point object which is configured to be an entry access point and at least one Access Point that is configured to be an exit access point. Each Access Point object which is part of the Access Zone must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

Properties in the Access Zone are written by the corresponding Access Point object when a valid credential is processed. The vendor must configure the IUT to have a sufficient number of valid credentials for the test being executed. Each credential must have an associated Access Credential object. Each Access Credential must have an associated Access Rights object that provides valid access to the zone.

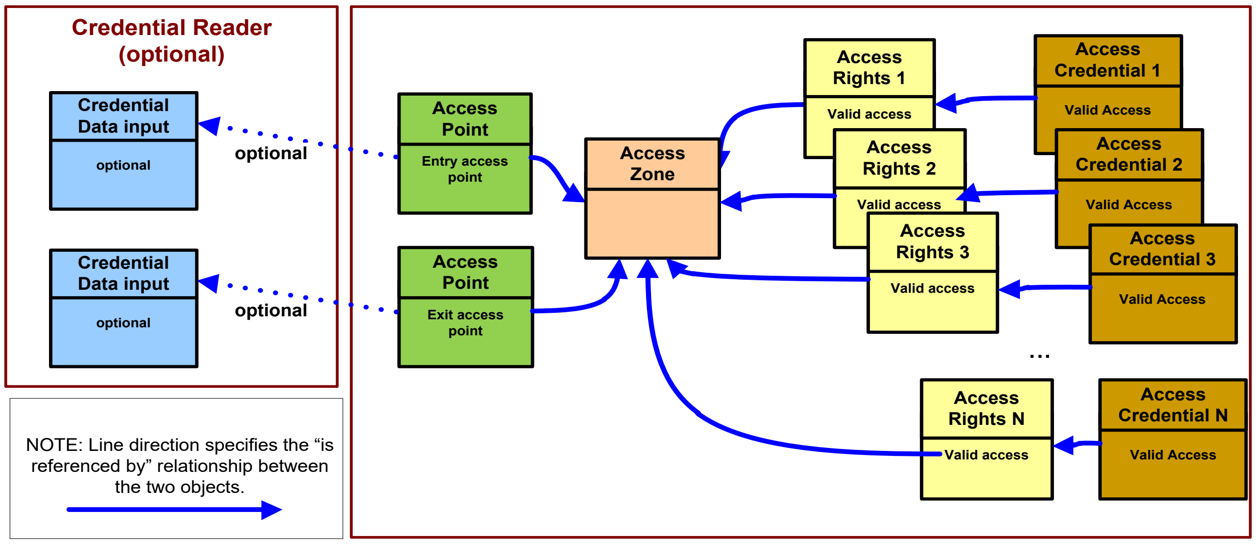

Figure X2: Objects used for testing the Access Zone object

7.3.2.X57.1 Occupancy State Test

Purpose: This test verifies that the occupancy state reflects the occupancy conditions of the zone.

Test Concept:
Adjust the occupancy count of the zone to levels above, at and below the Occupancy_Upper_Limit and Occupancy_Lower_Limit and verify that the Occupancy_State has the appropriate value.

Configuration Requirements:
See 7.3.2.X57. This test requires the following additional configuration:

a) The Access Zone shall not be out of service.
b) Occupancy_Count_Enable shall have a value of TRUE
c) The Occupancy_Upper_Limit shall have a value X where X>0.
d) If the property is supported, the Occupancy_Lower_Limit shall have a value Y where 0 < Y < X.

Test Steps:

-- test normal case where occupancy count is between the upper and lower limit

1. WRITE Adjust_Value = 0
2. VERIFY Occupancy_Count = 0
3. WRITE Adjust_Value = X-1
4. VERIFY Occupancy_Count = (X-1)
5. VERIFY Occupancy_State = NORMAL

-- verify the case where occupancy count is at and above the upper limit

6. WRITE Adjust_Value = 1
7. VERIFY Occupancy_Count = X
8. VERIFY Occupancy_State = AT_UPPER_LIMIT
9. WRITE Adjust_Value = 1
10. VERIFY Occupancy_Count = X+1
11. VERIFY Occupancy_State = ABOVE_UPPER_LIMIT

-- verify the case where occupancy count is at and above the lower limit

12. WRITE Adjust_Value = 0
13. VERIFY Occupancy_Count = 0
14. VERIFY Occupancy_State = BELOW_LOWER_LIMIT
15. WRITE Adjust_Value = Y
16. VERIFY Occupancy_Count = Y
17. VERIFY Occupancy_State = AT_LOWER_LIMIT

18. WRITE Adjust_Value = 1
19. VERIFY Occupancy_Count = (Y+1)
20. VERIFY Occupancy_State = NORMAL

-- verify occupancy state when occupancy counting is disabled
21. WRITE Occupancy_Count_Enable = FALSE
22. VERIFY Occupancy_State = DISABLED

7.3.2.X57.2 Occupancy Counting Test

Purpose: To verify that the occupancy counting functionality

Test Concept:
Present a credential at the entry and exit access points and test that the occupancy count is properly changed.

Configuration Requirements:
See 7.3.2.X57. This test requires the following additional configuration:

a) The Access Zone shall not be out of service.
b) Occupancy_Count_Enable shall have a value of TRUE
c) An access point which is an entry point to this zone shall be represented by Access Point object AP1. The Occupancy_Count_Adjust property for this object shall have a value of TRUE.
d) An access point which is an exit point to this zone shall be represented by Access Point object AP2. The Occupancy_Count_Adjust property for this object shall have a value of TRUE.
e) Two active credentials with valid access rights for the access points AP1 and AP2 shall be represented by Access Credential objects C1 and C2.

Test Steps:

-- verify that presenting a credential at the entry access point increases the occupancy count
1. WRITE Adjust_Value = 0
2. VERIFY Occupancy_Count = 0
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY Occupancy_Count = 1
5. MAKE (present credential C2 at credential reader for access point AP1)
6. VERIFY Occupancy_Count = 2

-- verify that presenting a credential at the exit access point decreases the occupancy count
7. MAKE (present credential C1 at credential reader for access point AP2)
8. VERIFY Occupancy_Count = 1
9. MAKE (present credential C2 at credential reader for access point AP2)
10. VERIFY Occupancy_Count = 0

7.3.2.X57.3 Keeping Track of Credentials Test

Purpose: To verify that the zone can keep track of the current credentials in the zone.

Test Concept:
Present a valid credential at an entry access point to this access zone. The object reference of the credential should be in the credentials in zone list. When the credential is presented to an exit access point for this zone the reference to the credential in the credential in zone list should be removed.

Configuration Requirements:
See 7.3.2.X57. This test requires the following additional configuration:

a) The Access Zone shall not be out of service.
b) The Credentials_In_Zone property shall be an empty list.
c) An access point which is an entry point to this zone shall be represented by Access Point object AP1.
d) An access point which is an exit point to this zone shall be represented by Access Point object AP2.
e) Two active credential with valid access rights for the access points AP1 and AP2 shall be represented by Access Credential object C1 and C2.

Test Steps:

-- verify that presenting a credential at an entry access point puts it in the Credentials_In_Zone list
1. VERIFY Credentials_In_Zone = ( )
2. MAKE (present credential C1 at credential reader for access point AP1)
3. VERIFY Credentials_In_Zone = (C1)
4. VERIFY Last_Credential_Added = C1
5. VERIFY Last_Credential_Added_Time = (the time that credential C1 was presented at AP1)
6. MAKE (present credential C2 at credential reader for access point AP1)
7. VERIFY Credentials_In_Zone = (C1,C2)
8. VERIFY Last_Credential_Added = C2
9. VERIFY Last_Credential_Added_Time = (the time that credential C2 was presented at AP1)

-- verify that presenting a credential at an exit access point removes it from the Credentials_In_Zone list
10. MAKE (present credential C1 at credential reader for access point AP2)
11. VERIFY Credentials_In_Zone = (C2)
12. VERIFY Last_Credential_Removed = C1
13. VERIFY Last_Credential_Removed_Time = the time that credential C1 was presented at AP2
14. MAKE (present credential C2 at credential reader for access point AP2)
15. VERIFY Credentials_In_Zone = ( )
16. VERIFY Last_Credential_Removed = C2
17. VERIFY Last_Credential_Removed_Time = the time that credential C2 was presented at AP2

7.3.2.X57.4 Passback Mode Test

Purpose: To verify the passback functionality and passback exemption. .

Test Concept: A valid credential is presented at the entry access point to this access zone. When the credential is presented a second time a passback notification should be generated and if the passback mode is set to hard passback then access to the zone should be denied. If the credential has a passback exemption then access will never be denied due to a passback violation.

Configuration Requirements:
See 7.3.2.X57. This test requires the following additional configuration:

a) The Access Zone shall not be out of service.
b) An access point which is an entry point to this zone shall be represented by Access Point object AP1. The Authoriztion_Mode property of AP1 shall have the value Authorize.
c) An access point which is an exit point to this zone shall be represented by Access Point object AP2. The Authoriztion_Mode property of AP2 shall have the value Authorize.
d) An active credential with valid access rights for the access point AP1 and AP2 shall be represented by Access Credential object C1.
e) The C1.Authorization_Exemptions list shall be empty.

Test Steps:

-- verify soft passback mode
1. MAKE (Passback_Mode = SOFT_PASSBACK)
2. READ EventTag = AP1,Access_Event_Tag
3. MAKE (present credential C1 at credential reader for access point AP1)
4. VERIFY AP1, Access_Event = GRANTED
5. VERIFY AP1,Access_Event_Time = (the time that credential C1 was presented)
6. VERIFY AP1,Access_Event_Credential = C1
7. VERIFY AP1,Access_Event_Tag = (EventTag + 1)
8. MAKE (present credential C1 at credential reader for access point AP1)
9. VERIFY AP1,Access_Event = GRANTED
10. VERIFY (AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
11. VERIFY AP1.Access_Event_Credential = C1
12. VERIFY AP1,Access_Event_Tag = (EventTag +2)
13. MAKE (present credential C1 at credential reader for access point AP2)

-- verify hard passback mode

14. MAKE (Passback_Mode = HARD_PASSBACK)
15. READ EventTag = AP1,Access_Event_Tag
16. MAKE (present credential C1 at credential reader for access point AP1)
17. VERIFY AP1.Access_Event = GRANTED
18. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
19. VERIFY AP1.Access_Event_Credential = C1
20. VERIFY AP1,Access_Event_Tag = EventTag + 1
21. MAKE (present credential C1 at credential reader for access point AP1)
22. VERIFY AP1.Access_Event = DENIED_PASSBACK
23. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
24. VERIFY AP1.Access_Event_Credential = C1
25. VERIFY AP1,Access_Event_Tag = (EventTag + 2)

-- verify passback timeout

26. WAIT (Passback_Timeout)
27. MAKE (present credential C1 at credential reader for access point AP1)
28. VERIFY AP1.Access_Event = GRANTED
29. VERIFY (AP1.Access_Event_Time = the time that credential C1 was presented)
30. VERIFY (AP1.Access_Event_Credential = C1)

-- verify hard passback off

31. MAKE (Passback_Mode = PASSBACK_OFF)
32. READ EventTag = AP1,Access_Event_Tag
33. MAKE (present credential C1 at credential reader for access point AP1)
34. VERIFY AP1.Access_Event = GRANTED
35. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
36. VERIFY AP1.Access_Event_Credential = C1
37. VERIFY AP1,Access_Event_Tag = EventTag + 1
38. MAKE (present credential C1 at credential reader for access point AP1)
39. VERIFY AP1.Access_Event = GRANTED
40. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
41. VERIFY (AP1.Access_Event_Credential = C1
42. VERIFY AP1,Access_Event_Tag = (EventTag + 2)

-- verify passback exemption

43. MAKE (Passback_Mode = HARD_PASSBACK)
44. MAKE (C1, Authorization_Exemption = (PASSBACK))
45. READ EventTag = AP1,Access_Event_Tag
46. MAKE (present credential C1 at credential reader for access point AP1)
47. VERIFY AP1.Access_Event = GRANTED
48. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
49. VERIFY AP1.Access_Event_Credential = C1
50. VERIFY AP1,Access_Event_Tag = EventTag + 1
51. MAKE (present credential C1 at credential reader for access point AP1)
52. VERIFY AP1.Access_Event = GRANTED
53. VERIFY AP1.Access_Event_Time = (the time that credential C1 was presented most recently)
54. VERIFY AP1.Access_Event_Credential = C1
55. VERIFY AP1,Access_Event_Tag = (EventTag + 2)

[In BTL Checklist, modify the existing Access User object type to Section 3, Objects]

Support	Listing	Option
Access User Object		
	R[+]	Base Requirements
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
	[+]~~Contact BTL for interim tests for this object.~~	

[In BTL Test Plan, modify the existing Access User object tests in section 3.46]

3.46 Access User Object

3.46.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *There are no base requirements tests for this section.*

3.46.2 Contains an object with Reliability_Evaluation_Inhibit Property

The IUT contains, or can be made to contain, a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	

[In BTL Checklist, modify the existing Access Rights object type to Section 3, Objects]

Support	**Listing**	**Option**
Access Rights Object		
	R+	Base Requirements
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
	+~~Contact BTL for interim tests for this object.~~	

[In BTL Test Plan, Modify the existing Access Rights Object section]

3.47 Access Rights Object

3.47.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *Base requirements must be met by any IUT that can contain Access Rights objects*

BTL - 7.3.2.X59.1 - Enable Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X59.2 - Negative Rules Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X59.3 - Positive Access Rules Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X59.4 - Accompaniment Test		
	Test Conditionality	*If the IUT does not support the Accompaniment_Time in Access Point Object Type, this test shall be skipped.*
	Test Directives	
	Testing Hints	

3.47.2 Contains an object with Reliability_Evaluation_Inhibit Property

The IUT contains, or can be made to contain, a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	

[In BTL Specified Tests, add Access Rights object specific tests in section 7.3.2.59.X]

7.3.2.59 Access Rights Object Tests

Many of the tests for the Access Rights object require interactions from other BACnet access control objects as a result of a credential being processed. The required and optional BACnet object types are shown in figure d.

The Access Rights object specifies both positive and negative access rights. This list of access rights is used by the Access Point object to determine the access decision. To test the access rights the vendor must configure the IUT to have at least one Access Point object which is referenced in the Access Rights objects used for this test.

Each Access Point object used in the test must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

The vendor must configure the IUT such that for each Access Rights object there is at least one corresponding Access Credential object that references the Access Rights object.

When the access decision is determined the IUT shall provide a method to show the result. Typically the decision is exposed through the Access Door object. When access is granted the door is pulsed unlocked and when denied the door remains locked. If the Access Door object is not used then another method of showing the result shall be configured.

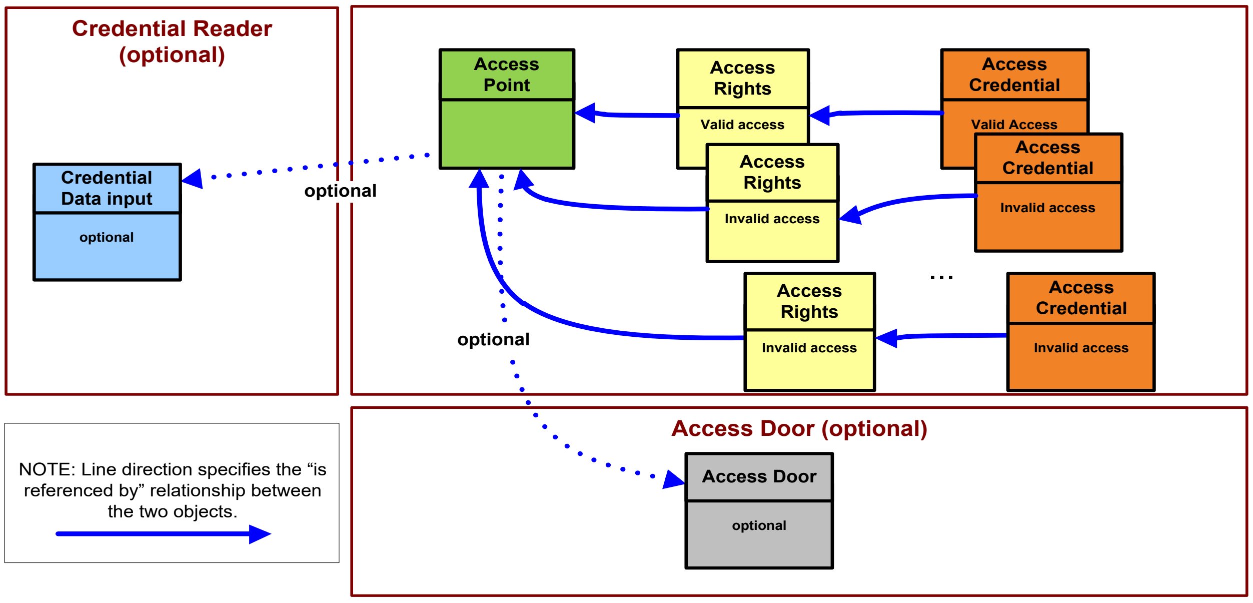

Figure d: Objects used for testing the Access Rights object

7.3.2.X59.1 Enable Test

Purpose: This test verifies that the access rights object does not allow access when the Enable property is FALSE.

Test Concept: Present a valid credential at the access point. Since the access rights object that provides the access rights to the access point is not enabled access shall be denied.

Configuration Requirements:
See 7.3.2.X59. This test requires the following additional configuration:

a) An access point shall be represented by Access Point object AP1.
b) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
c) AR1 shall have the Enable property set to TRUE
d) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

-- verify access granted with this access rights object when enable property is TRUE

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1

-- verify access denied with this access rights object when Enable property is FALSE

5. WRITE AR1, Enable = FALSE
6. MAKE (present credential C1 at credential reader for access point AP1)
7. VERIFY AP1,Access_Event = DENIED_NO_ACCESS_RIGHTS
8. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
9. VERIFY AP1,Access_Event_Credential = C1

7.3.2.X59.2 Negative Rules Test

Configuration Requirements:
See 7.3.2.X59. This test requires the following additional configuration:

a) An access point shall be represented by Access Point object AP1.
b) An access rights object which specifies negative access rights to AP1 shall be represented by Access Rights object AR1.
c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = DENIED_POINT_NO_ACCESS_RIGHTS
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1

7.3.2.X59.3 Positive Access Rules Test

Purpose: To verify that the positive access rules explicitly enable access to an access point.

Test Concept: Present a credential at the access point. The access point at which the credential is trying to get entry is in the positive rules list and that rule is valid at the current time. Access to the access point is granted.

Configuration Requirements:
See 7.3.2.X59. This test requires the following additional configuration:

a) An access point shall be represented by Access Point object AP1.
b) An access rights object which specifies positive access rights to AP1 shall be represented by Access Rights object AR1.
c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1

7.3.2.X59.4 Accompaniment Test

Purpose: To verify that the accompaniment functionality works properly.

Test Concept: Present a credential which needs accompaniment at the access point. Wait for the accompaniment time, as specified in the access point. When this times out the credential should be denied entry to the access point. Present the first credential again at the access point. Present a second credential at the access point which has the rights to accompany the first credential. Access should be granted to the first credential.

Configuration Requirements:
See 7.3.2.X59. This test requires the following additional configuration:

a) An access point shall be represented by Access Point object AP1. If the Accompaniment_Time property is supported it shall be set to a value > 0.
b) An access rights object which specifies positive access rights to AP1 shall be represented by Access Rights object AR1.
c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1 that requires accompaniment.
d) An active credential which has access rights which allow it to accompany a credential with access rights specified by AR1 through AP1, shall be represented by Access Credential object C2.
e) An active credential which has valid access rights to AP1 but which does not meet the accompaniment requires of AR1, shall be represented by Access Credential object C3.

Test Steps:

-- valid access through the access point
1. READ Tag = Access_Event_Tag
2. MAKE (present credential C1 at credential reader for access point AP1)
3. MAKE (present credential C2 at credential reader for access point AP1)
4. VERIFY AP1,Access_Event = GRANTED
5. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
6. VERIFY AP1,Access_Event_Credential = C1
7. VERIFY AP1,Access_Event_Tag = (Tag + 1)

-- no accompaniment presented
8. MAKE (present credential C1 at credential reader for access point AP1)
9. WAIT AP1,Accompaniment_Time
10. VERIFY AP1,Access_Event = DENIED_NO_ACCOMPANIMENT
11. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
12. VERIFY AP1,Access_Event_Credential = C1
13. VERIFY Access_Event_Tag = (Tag + 2)

-- Invalid accompaniment
14. MAKE (present credential C1 at credential reader for access point AP1)
15. MAKE (present credential C3 at credential reader for access point AP1)
16. VERIFY AP1,Access_Event = DENIED_INCORRECT_ACCOMPANIMENT
17. VERIFY AP1,Access_Event_Time = (the time that the credential C1 was presented)
18. VERIFY AP1,Access_Event_Credential = C1
19. VERIFY AP1,Access_Event_Tag = (Tag + 3)

[In BTL Checklist, Update the existing Access Credential Object Section]

Support	Listing	Option
Access Credential Object		
	R~~[1]~~	Base Requirements
	O	*Supports Days_Remaining*
	O	*Supports Absentee_Limit*
	O	*Supports Last_Access_Point, Last_Use_Time and Last_Access_Event*
	O	*Supports Writable Extended_Time_Enabled*
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
	~~[1]Contact BTL for interim tests for this object.~~	

[In BTL Test Plan, Modify the existing Access Credential Object section]

3.48 Access Credential Object

3.48.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *Base requirements must be met by any IUT that can contain Access Credential objects*

BTL - 7.3.2.X60.1 - Credential Status, Credential Disable and Reason for Disable Test		
	Test Conditionality	*If the IUT does not support certain reasons for disable then the corresponding steps shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X60.2 - Activation Time and Expiration Time Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X60.3 - Disabled Access Rights Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.48.2 Supports Days_Remaining

The IUT supports the Access Credential Objects with Days_Remaining property

BTL - 7.3.2.X60.4 - Days Remaining and Uses Remaining Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.48.3 Supports Absentee_Limit

The IUT supports the Access Credential Objects with Absentee_Limit property

BTL - 7.3.2.X60.5 - Absentee Limit Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.48.4 Supports Last_Access_Point, Last_Use_Time and Last_Access_Event

The IUT supports the Access Credential Objects with Last_Access_Point, Last_Use_Time and Last_Access_Event properties

BTL - 7.3.2.X60.6 - Last Access Point, Last Use Time and Last Access Event Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.48.5Supports Writable Extended_Time_Enable

The IUT supports the Access Credential Objects with writable Extended_Time_Enable property

BTL - 7.3.2.X60.7 - Extended Time Enable Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.48.6 Contains an object with Reliability_Evaluation_Inhibit Property

The IUT contains, or can be made to contain, a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	

[In BTL Specified Tests, add Access Credential object specific tests in section 7.3.2.60X]

7.3.2.X60 Access Credential Object Tests

Many of the tests for the Access Credential object require interactions from other BACnet access control objects as a result of a credential being processed. The required and optional BACnet object types are shown in figure e.

The Access Credential defines a list of credential values (authentication factors) that are used to identify the credential. The credential values are used by the Access Point object in determining the access control decision. To test the Access Credential object the vendor must configure the IUT to have at least one Access Point object. Each Access Credential used in this test must reference an Access Rights object which references the Access Point object

Each Access Point object used in the test must have an associated Credential Data Input or a proprietary method to input credential values to the Access Point.

The vendor must configure the IUT such that for each Access Rights object there is at least one corresponding Access Credential object that references the Access Rights object.

When the access decision is determined the IUT shall provide a method to show the result. Typically the decision is exposed through the Access Door object. When access is granted the door is pulsed unlocked and when denied the door remains locked. If the Access Door object is not used then another method of showing the result shall be configured.

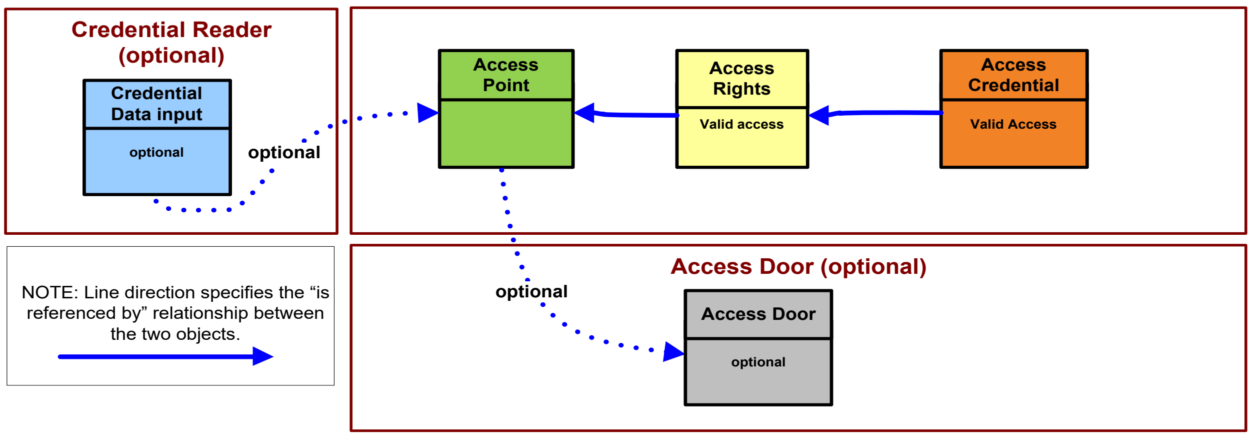

Figure e: BACnet Objects used for testing the Access Credential object

7.3.2.X60.1 Credential Status, Credential Disable and Reason for Disable Test

Purpose: To verify the ability to disable the credential and set the associated reason and to enable the credential.

Test Concept: The credential status is set to INACTIVE and the corresponding reason or reasons are written to the Reason_For_Disable list.

Configuration Requirements:
See 7.3.2.X60. This test requires the following additional configuration:

a) An access credential shall be represented by Access Credential C1.
b) The Credential_Status property shall have the value ACTIVE.
c) The Reason_For_Disable property shall be empty.

Note: This tests only verifies the most common disable reasons (DISABLED_MANUAL, DISABLED_NOT_YET_ACTIVE, DISABLED_EXPIRED).

Test Steps:

```
-- test DISABLED_MANUAL
1.  VERIFY  Credential_Status  = ACTIVE
2.  VERIFY Reason_For_Disable = ( )
3.  MAKE (add DISABLED_MANUAL to Reason_For_Disable property)
4.  VERIFY Reason_For_Disable = (DISABLED_MANUAL)
5.  VERIFY Credential_Status = INACTIVE
6.  MAKE (remove DISABLED_MANUAL from Reason_For_Disable property)
7.  VERIFY Reason_For_Disable = ( )
8.  VERIFY  Credential_Status  = ACTIVE

-- test DISABLED_NOT_YET_ACTIVE
9.  VERIFY  Credential_Status  = ACTIVE
10. VERIFY Reason_For_Disable = ( )
11. MAKE (set Activation_Time = time > current time)
12. VERIFY Reason_For_Disable = (DISABLED_NOT_YET_ACTIVE)
13. VERIFY Credential_Status = INACTIVE
14. MAKE (set Activation_Time = time < current time)
15. VERIFY Reason_For_Disable = ( )
16. VERIFY  Credential_Status  = ACTIVE

-- test DISABLED_EXPIRED
```

17. VERIFY Credential_Status = ACTIVE
18. VERIFY Reason_For_Disable = ( )
19. MAKE (set Expiration_Time = time < current time)
20. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
21. VERIFY Credential_Status = INACTIVE
22. MAKE (set Expiration_Time = time > current time)
23. VERIFY Reason_For_Disable = ( )
24. VERIFY Credential_Status = ACTIVE

7.3.2.X60.2 Activation Time and Expiration Time Test

Purpose: To test the activation time and expiration time functionality of this object.

Test Concept: The Activation_Time of the credential is set to a time in the future and the credential should be disabled. The Expiration_Time is set to a time in the past and the credential should be disabled.

Configuration Requirements:
See 7.3.2.X60. This test requires the following additional configuration:

a) The Credential_Status property shall have the value ACTIVE.
b) The Reason_For_Disable property shall be empty.
c) The Activation_Time shall have an initial value of 0xFF
d) The Expiration_Time shall have an initial value of 0xFF.

Test Steps:

-- test activation time
1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ( )
3. MAKE (set Activation_Time = time > current time)
4. VERIFY Credential_Status = INACTIVE
5. VERIFY Reason_For_Disable = (DISABLED_NOT_YET_ACTIVE)
6. MAKE (set Activation_Time = time < current time)
7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ( )

-- test expiration time
9. VERIFY Credential_Status = ACTIVE
10. VERIFY Reason_For_Disable = ( )
11. MAKE (Expiration_Time = time < current time)
12. VERIFY Credential_Status = INACTIVE
13. VERIFY Reason_For_Disable = (DISABLED_EXPIRED)
14. MAKE (Expiration_Time = time > current time)
15. VERIFY Credential_Status = ACTIVE
16. VERIFY Reason_For_Disable = ( )

7.3.2.X60.3 Disabled Access Rights Test

Purpose: To verify the enable field disables an access right for this credential when set to FALSE.

Configuration Requirements:
See 7.3.2.X60. This test requires the following additional configuration:

a) An access point shall be represented by Access Point object AP1.
b) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
c) An active credential where Assigned_Access_Rights[1].Assigned-Access-Rights = AR1 shall be represented by Access Credential object C1. Assigned_Access_Rights[1].Enable shall be TRUE.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)

4. VERIFY AP1,Access_Event_Credential = C1
5. WRITE Assigned_Access_Rights[1].Enable = FALSE
6. MAKE (present credential C1 at credential reader for access point AP1)
7. VERIFY AP1,Access_Event = DENIED_NO_ACCESS_RIGHTS
8. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
9. VERIFY AP1,Access_Event_Credential = C1

7.3.2.X60.4 Days Remaining and Uses Remaining Test

Purpose: To test the days remaining and uses remaining functionality of this object.

Test Concept:
Set the Days_Remaining property to 0. The credential will become inactive and the corresponding reason for disable put in the Reason_For_Disable property.

Set the Uses_Remaining to 0. The credential will become inactive and the corresponding reason for disable will be put in the Reason_For_Disable property.

Configuration Requirements:
See 7.3.2.X60. This test requires the following additional configuration:
a) The Credential_Status property shall have the value ACTIVE.
b) The Reason_For_Disable property shall be empty.
c) Days_Remaining shall have a value of -1 or >0.
d) Uses_Remaining shall have a value of -1 or >0.

Test Steps:

-- test day remaining
1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ( )
3. MAKE (Days_Remaining = 0)
4. VERIFY Credential_Status = INACTIVE
5. VERIFY Reason_For_Disable = (DISABLED_MAX_DAYS)
6. MAKE (Days_Remaining = -1 OR Days_Remaining > 0)
7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ( )

-- test uses remaining
9. VERIFY Credential_Status = ACTIVE
10. VERIFY Reason_For_Disable = ( )
11. MAKE (Uses_Remaining = 0)
12. VERIFY Credential_Status = INACTIVE
13. VERIFY Reason_For_Disable = (DISABLED_MAX_USES)
14. MAKE (Uses _Remaining = -1 OR Uses _Remaining > 0)
15. VERIFY Credential_Status = ACTIVE
16. VERIFY Reason_For_Disable = ( )

7.3.2.X60.5 Absentee Limit Test

Purpose: To verify the absentee limit functionality of this object.

Test Concept: Set the Absentee_Limit property some value >= 0. Use the credential to access an access point. Change the current date to one that is greater than (current date + Absentee_Limit) and attempt to gain access to an access point. Access should be denied because the credential should be disabled due to inactivity.

Configuration Requirements:
See 7.3.2.X60. This test requires the following additional configuration:
a) Absentee_Limit >= 0
b) The Credential_Status property shall have the value ACTIVE.

c) The Reason_For_Disable property shall be empty.
d) Days_Remaining shall have a value > 0.
e) Last_Use_Time shall be set to a valid date and time.

Test Steps:

1. VERIFY Credential_Status = ACTIVE
2. VERIFY Reason_For_Disable = ( )
3. TRANSMIT UTCTimeSynchronization-Request, 'Time' = (any day and time greater than Absentee_Limit days)
4. VERIFY Credential_Status = INACTIVE
5. VERIFY Reason_For_Disable = (DISABLED_INACTIVITY)

- clear the condition
6. MAKE (set Absentee_Limit > number of days since Last_Use_Time)
7. VERIFY Credential_Status = ACTIVE
8. VERIFY Reason_For_Disable = ( )

7.3.2.X60.6 Last Access Point, Last Use Time and Last Access Event Test

Purpose: To verify that the Last Access Point, Last Access Event and Last Use Time properties are updated when this credential is used at an access point.

Configuration Requirements:
See 7.3.2.X60. This test requires the following additional configuration:
a) An access point shall be represented by Access Point object AP1.
b) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
c) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.

Test Steps:

1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1
5. VERIFY Last_Access_Point = AP1
6. VERIFY Last_Use_Time = AP1,Access_Event_Time
7. VERIFY Last_Access_Event = GRANTED

7.3.2.X60.7 Extended Time Enable Test

Purpose: To verify that a credential used at an access point with the extended flag set to TRUE results in the corresponding door to pulse open for a Pulse_Extended period of time as defined in the door object.

Configuration Requirements:
See 7.3.2.X60. This test requires the following additional configuration:
a) An access door shall be represented by Access Door object AD1 and Door_Delay_Time shall be 0.
b) An access point, which controls AD1, shall be represented by Access Point object AP1.
c) An access rights object which specifies valid access rights to AP1 shall be represented by Access Rights object AR1.
d) An active credential with access rights specified by AR1 shall be represented by Access Credential object C1.
e) The Extended_Time_Enable property shall be FALSE.

Test Steps:

-- verify that PULSE_UNLOCK is written when the extended time enable is FALSE
1. MAKE (present credential C1 at credential reader for access point AP1)
2. VERIFY AP1,Access_Event = GRANTED
3. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
4. VERIFY AP1,Access_Event_Credential = C1
5. BEFORE Door_Pulse_Time VERIFY AD1, Present_Value = PULSE_UNLOCK

-- verify that EXTENDED_PULSE_UNLOCK is written when the extended time enable is TRUE

6. WRITE Extended_Time_Enable = TRUE
7. MAKE (present credential C1 at credential reader for access point AP1)
8. VERIFY AP1,Access_Event = GRANTED
9. VERIFY AP1,Access_Event_Time = (the time that the credential was presented)
10. VERIFY AP1,Access_Event_Credential = C1
11. BEFORE Door_Pulse_Time VERIFY AD1, Present_Value = EXTENDED_PULSE_UNLOCK

[In BTL Checklist, Modify existing Credential Data Input Object type to Section 3, Objects]

Support	Listing	Option
Credential Data Input Object		
	R+	Base Requirements
	O	*Supports writable Out_Of_Service properties*
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
	~~+Contact BTL for interim tests for this object.~~	

[In BTL Test Plan, Modify existing Credential Data Input Object section]

3.49 Credential Data Input Object

3.49.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *Base requirements must be met by any IUT that can contain Credential Data Input objects*

BTL - 7.3.2.X61.1 - Return From Out Of Service Undefined Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X61.2 - Read Valid Authentication Factor Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.49.2 Supports Writable Out_Of_Service Properties

The Out_Of_Service property in Credential Data Input objects contained in the IUT are writable.

135.1-2013 - 7.3.1.1 - Out_Of_Service, Status_Flags, and Reliability Tests		
	Test Conditionality	*Must be executed.*
	Test Directives	*The test shall be executed using a Credential Data Input object*
	Testing Hints	

3.49.3 Contains an object with Reliability_Evaluation_Inhibit Property

The IUT contains, or can be made to contain, a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	

[In BTL Specified Tests, add Credential Data Input object specific tests in section 7.3.2.X61]

7.3.2.X61 Credential Data Input Object Tests

The Credential Data Input object type represents a device or process that reads an authentication factor from a physical device such as a card reader, key pad or biometric device. There are countless variations and authentication formats supported for these devices. As such, there is not a standard format or device configuration that can be mandated for these tests.

The vendor must configure the IUT such that the Credential Data Input device can read an authentication factor from the corresponding physical device including setting the Supported_Formats[1] property to the correct authentication factor format (figure f). This configuration is considered to be a local matter and will not be tested.

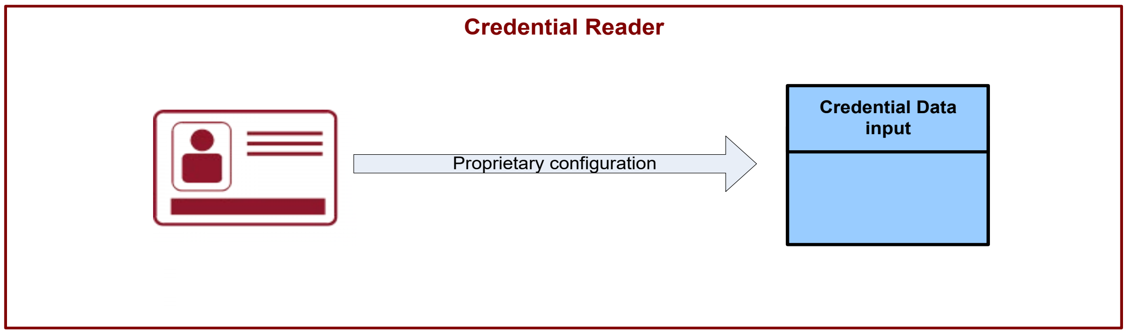

Figure f: Credential Data Input configuration

7.3.2.X61.1 Return from Out Of Service Undefined test

Purpose: To verify that the Present_Value. Format-Type becomes undefined when out of service is set to false.

Configuration Requirements:
See 7.3.2.X61. This test requires the following additional configuration:

a) The Out_Of_Service property shall be TRUE.

Test Steps:

1. WRITE Out_Of_Service = FALSE
2. VERIFY Present_Value. Format-Type = UNDEFINED
3. VERIFY Present_Value. Format-Class = 0

7.3.2.X61.2 Read Valid Authentication Factor Test

Purpose: To verify that Present_Value is set to the proper value when an authentication factor with a recognized format is read at the corresponding physical device.

Configuration Requirements:
See 7.3.2.X61. This test requires the following additional configuration:

a) The Out_Of_Service property shall be FALSE.
b) Two authentication factors, AF1 and AF2, shall be provided which can be read by the physical device which have the format specified in Supported_Formats[1].

Test Steps:

-- test AF1
1. MAKE (present AF1 at the credential reader)
2. VERIFY Present_Value. Format-Type = Supported_Formats[1]. Format-Type
3. VERIFY Present_Value. Format-Class = Supported_Formats[1]. Format-Class
4. VERIFY Present_Value. Value = the authentication format value of AF1

-- test AF2
5. MAKE (present AF2 at the credential reader)
6. VERIFY Present_Value. Format-Type = Supported_Formats[1]. Format-Type

7. VERIFY Present_Value. Format-Class = Supported_Formats[1]. Format-Class
8. VERIFY Present_Value. Value = the authentication format value of AF2

BTL-15.2m-2: Access Door Object Tests [BTLWG-476]

Overview:
To create new tests for Access Door (Protocol Rev 6 object type)

Changes:

[In BTL Checklist, modify existing Access Door object type to Section 3, Objects]

Support	Listing	Option
Access Door Object		
	R~~+~~	Base Requirements
	R	*Supports Command Prioritization*
	S	*Supports writable Out_Of_Service properties*
	C[1]	*Supports Door_Status*
	O	*Supports Lock_Status*
	O	*Supports Secured_Status*
	O	*Supports Door_Unlock_Delay_Time*
	O	*Supports Masked_Alarm_Values*
	O	*Supports Intrinsic Reporting*
	O	*Contains an object with Reliability_Evaluation_Inhibit Property*
	[1] *If Secured_Status is supported, this is required.*	
	~~+Contact BTL for interim tests for this object.~~	

Changes:

[In BTL Test Plan, modify existing Access Door object tests in section 3.42]

3.42 Access Door Object

3.42.1 Base Requirements

~~Contact BTL for interim tests for this object.~~ *Base requirements must be met by any IUT that supports Access Door objects*

BTL - 7.3.2.X55.1.X1 - Commandable Present_Value Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.42.2 Supports Command Prioritization

135.1-2013 - 7.3.1.2 - Relinquish Default Test		
	Test Conditionality	*If no object can be made to meet the configuration requirements, this test shall be skipped.*
	Test Directives	
	Testing Hints	
135.1-2013 - 7.3.1.3 - Command Prioritization Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.42.3 Supports Writable Out_Of_Service Properties

The IUT contains or can be made to contain writable Out_Of_Service property.

135.1-2013 - 7.3.1.1 - Out_Of_Service, Status_Flags, and Reliability Tests		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	
BTL - 7.3.2.X55.1.X2 - Door_Status, Lock_Status and Door_Alarm_State Tests		
	Test Conditionality	*If neither Door_Status, Lock_Status or Door_Alarm_State is supported, this test shall be skipped.*
	Test Directives	
	Testing Hints	

3.42.4 Supports Door_Status

The IUT contains or can be made to contain Door_Status property which is writable when Out_Of_Service is True.

BTL - 7.3.2.X55.1.X3 - Door_Status with physical door status Tests		
	Test Conditionality	*If the Door_Status property is permanently configured to have the value UNUSED then this test shall be skipped.*
	Test Directives	
	Testing Hints	

3.42.5 Supports Lock_Status

The IUT contains or can be made to contain Lock_Status property which is writable when Out_Of_Service is True.

BTL - 7.3.2.X55.1.X4 - Lock_Status Tests		
	Test Conditionality	*If the physical lock cannot be manipulated without writing to Present_Value of the associated Access Door objet then this test shall be skipped.*
	Test Directives	
	Testing Hints	

3.42.6 Supports Secured_Status

The IUT contains or can be made to contain Secured_Status property.

BTL - 7.3.2.X55.1.X5 - Secured_Status Tests		
	Test Conditionality	*If the Secured_Status property is permanently configured to have the value UNKNOWN then this test shall be omitted.*
	Test Directives	
	Testing Hints	

3.42.7 Supports Door_Unlock_Delay_Time

The IUT contains or can be made to contain a writable or read-only Door_Unlock_Delay_Time property

BTL - 7.3.2.X55.1.X6 - Door_Unlock_Delay_Time Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.42.8 Supports Masked_Alarm_Values

The IUT contains or can be made to contain Masked_Alarm_Value property.

BTL - 7.3.2.X55.1.X7 - Masked_Alarm_Values Test

	Test Conditionality	*If Out_Of_Service is not writeable and cannot be set to TRUE by any other means, this test shall be skipped.*
	Test Directives	
	Testing Hints	

3.42.9 Supports Intrinsic Reporting

The IUT supports intrinsic reporting.

BTL - 7.3.2.X55.1.X8 - Door_Open_Too_Long Test		
	Test Conditionality	*Must be executed.*
	Test Directives	
	Testing Hints	

3.42.10 Contains an object with Reliability_Evaluation_Inhibit Property

The IUT contains or can be made to contain a Reliability_Evaluation_Inhibit property that is configurable to a value of TRUE.

BTL - 7.3.1.X8.1 - Reliability_Evaluation_Inhibit Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	
BTL - 7.3.1.X8.2 - Reliability_Evaluation_Inhibit Summarization Test		
	Test Conditionality	*If no object exists in the IUT for which fault conditions can be generated then this test shall be skipped.*
	Test Directives	
	Testing Hints	

Changes:

[In BTL Specified Tests, add Access Door object specific tests in section 7.3.2.X]

7.3.2.X55.1.X1 Commandable Present_Value Test

Purpose: To verify that writing to the Present_Value will cause a corresponding change to the physical output.

Test Concept: The IUT shall be configured with a door control output that can be observed during the test. The Present_Value property is written with each of the following values: UNLOCK, LOCK, PULSE_UNLOCK, EXTENDED_PULSE_UNLOCK and the Access Door object is monitored to ensure that the door locks and unlocks appropriately.

Configuration Requirements: The Relinquish_Default shall have the value LOCK. All writes are at a priority higher than any internal algorithms writing to this property. Out_Of_Service shall be set to FALSE. Prior to the test the Present_Value shall have the value LOCK and the IUT is in a state that would cause the door to be locked.

Test Steps:

```
-- Test UNLOCK value
1.  WRITE Present_Value = UNLOCK
2.  WAIT (Internal Processing Fail Time)
3.  IF (Lock_Status is present) THEN
        VERIFY Lock_Status = UNLOCKED
4.  CHECK (that the door control output is in a state that would cause the door to be unlocked)

-- Test LOCK value
5.  WRITE Present_Value = LOCK
6.  WAIT (Internal Processing Fail Time)
7.  IF (Lock_Status is present) THEN
        VERIFY Lock_Status =    LOCKED
```

```
8.  CHECK (that the door control output  is in a state that would cause the door to be locked)

-- Test PULSE_UNLOCK value
9.  WRITE Present_Value = PULSE_UNLOCK
10. WAIT (Internal Processing Fail Time + Door_Unlock_Delay_Time if present)
11. IF (Lock_Status is present) THEN
        VERIFY Lock_Status = UNLOCKED
12. CHECK (that the IUT is in a state that would cause the door to be unlocked)
13. WAIT (Door_Pulse_Time)
14. VERIFY Present_Value = LOCK
15. IF (Lock_Status is present) THEN
        VERIFY Lock_Status = LOCKED
16. CHECK (that the door control output  is in a state that would cause the door to be locked)

-- Test EXTENDED_PULSE_UNLOCK value
17. WRITE Present_Value = EXTENDED_PULSE_UNLOCK
18. WAIT (Internal Processing Fail Time + Door_Unlock_Delay_Time if present)
```

7.3.2.X55.1.X2 Door_Status, Lock_Status and Door_Alarm_State Tests

Purpose: This test case verifies that Door_Status, Lock_Status and Door_Alarm_State properties are writable when Out_Of_Service is TRUE.

Test Concept: Set Out_Of_Service to TRUE and then make sure one at a time that Door_Status, Lock_Status and Door_Alarm_State, if present, are writable.

Configuration Requirements : If the Out_Of_Service property of this object is not writable, and if the Out_Of_Service property cannot be changed by other means, then this test shall be omitted. All writes to the Present_Value shall be performed at a priority higher (numerically smaller) than any internal algorithms writing to this property. For testing Door_Alarm_State, test only values listed in either the Alarm_Values or Fault_Values.

Test Steps:

```
1.  MAKE (Out_Of_Service TRUE)
2.  VERIFY Status_Flags = (?,?,?,TRUE)
3.  IF (Door_Status is present) THEN
        REPEAT X = (all values of the Door_Status enumeration values supported by the property)
            DO {
                WRITE Door_Status = X
                VERIFY Door_Status = X
            }
4.  IF (Lock_Status is present) THEN
        REPEAT X = (all values of the Lock_Status enumeration values supported by the property)
            DO {
                WRITE Lock_Status = X
                VERIFY Lock_Status = X
            }
5.  IF (Door_Alarm_State is present) THEN
        REPEAT X = (all values of the Door_Alarm_State enumeration values supported by the property)
            DO {
                WRITE Door_Alarm_State = X
                VERIFY Door_Alarm_State = X
            }
```

7.3.2.X55.1.X3 Door_Status with physical door status Tests

Purpose: To verify that the Door_Status property reflects the state of the physical door (CLOSED, OPENED, UNUSED and DOOR_FAULT if the object supports detecting door faults).

Test Concept: The IUT is configured to monitor the state of a physical door. The physical door may be represented by a BACnet input object or through some proprietary method.

Configuration Requirements: The IUT shall be configured such that it can determine the state of a door. The Access_Door object associated with this physical door shall be configured with Out_Of_Service = FALSE.

Test Steps:

```
1.  MAKE (set physical door to the closed state)
2.  VERIFY Door_Status = CLOSED
3.  MAKE (set physical door to the opened state)
4.  VERIFY Door_Status = OPENED
5.  IF (the object supports detecting door faults)
        MAKE (set the physical door to a state that would cause the Door_Status to take on a value of
                DOOR_FAULT)
        VERIFY Door_Status = DOOR_FAULT
6.  IF (possible to remove a door status input associated with the door)
        MAKE (remove a door status input associated with the door)
7.      VERIFY Door_Status = UNUSED | UNKNOWN
```

7.3.2.X55.1.X4 Lock_Status Tests

Purpose: To verify that the Lock_Status property reflects the state of the physical lock. (LOCKED, UNLOCKED and LOCK _FAULT if the object supports detecting lock faults).

Test Concept: The IUT monitors the state of a physical lock. The state of the physical lock may be represented by a BACnet input object or through some proprietary method.

Configuration Requirements: The IUT shall be configured such that it can monitor the state of the physical lock. The Access_Door object associated with this physical door shall be configured with Out_Of_Service = FALSE. The physical lock shall be manipulated other than through the Access Door object.

Note to tester: The physical lock shall be manipulated other than through the Access Door object.

Test Steps:

```
1.  MAKE (set the physical lock to a state that would cause the Lock_Status to take on a value of
        LOCKED)
2.  VERIFY Lock_Status = LOCKED
3.  MAKE (set the physical lock to a state that would cause the Lock_Status to take on a value of
        UNLOCKED)
4.  VERIFY Lock_Status = UNLOCKED
5.  IF (the object and the lock support detecting lock faults)
        MAKE (set the physical lock to a state that would cause the Lock_Status to take on a value of
                LOCK_FAULT)
        VERIFY Lock_Status = LOCK_FAULT
```

7.3.2.X55.1.X5 Secured_Status Tests

Purpose: To verify that the Secured_Status property reflects the state of the physical lock, the physical door and the state of the Access Door object.

Test Concept: Start the test by creating a condition where the Secured_Status = SECURED. Then create various conditions one at a time to verify that the Secured_Status becomes UNSECURED when it should.

Configuration Requirements: All writes to the Present_Value shall be performed at a priority higher than any internal algorithms writing to this property. If this object supports intrinsic reporting then the Alarm_Values property shall be empty. If this object supports the Masked_Alarm_Values property then it shall be empty. Out_Of_Service is FALSE.

Test Steps:

```
-- Create a condition where the Secured_Status becomes SECURED
1.  WRITE Present_Value = LOCK
```

```
2.  WAIT (**Internal Processing Fail Time**)
3.  VERIFY Status_Flags = (FALSE ?,?, ?)
4.  IF (Lock_Status property is present)
        MAKE (Lock_Status = LOCKED or UNUSED)
5.  MAKE (Door_Status =  CLOSED or UNUSED)

-- Verify that the Secured_Status is SECURED when it should
6.  VERIFY Secured_Status = SECURED

-- Verify that Secured_Status is UNSECURED when Present_Value is anything other than LOCKED
7.  REPEAT X = (UNLOCK, PULSE_UNLOCK, EXTENDED_PULSE_UNLOCK) DO {
        WRITE Present_Value = X
        WAIT (Internal Processing Fail Time)
        VERIFY Secured_Status = UNSECURED
    }
-- Recreate a condition where the Secured_Status becomes SECURED again
8.  WRITE Present_Value = LOCK
9.  WAIT (Internal Processing Fail Time)
10. VERIFY Secured_Status = SECURED

-- Verify that Secured_Status is UNSECURED when Masked_Alarm_Value, if exist, is NOT empty
11. IF (Masked_Alarm_Values is present) THEN
        MAKE (Masked_Alarm_Values = (any valid BACnetDoorAlarmState enumeration))
        WAIT(Internal Processing Fail Time)
        VERIFY Secured_Status = UNSECURED

-- Recreate a condition where the Secured_Status becomes SECURED again
        MAKE ( Masked_Alarm_Values = {})
        WAIT (Internal Processing Fail Time)
        VERIFYSecured_Status = SECURED

-- Verify that Secured_Status is UNSECURED when Lock_Status, if present, is anything other than LOCKED or UNUSED
12. IF (Lock_Status property is present) THEN
        REPEAT X = (UNLOCKED. UNKNOWN, LOCK_FAULT) DO {
            MAKE (Lock_Status = X)
            WAIT (Internal Processing Fail Time)
            VERIFY Secured_Status = UNSECURED
        }
        REPEAT X = (LOCKED, UNUSED) DO {
            MAKE (Lock_Status = X)
            VERIFY Secured_Status = SECURED
        }

-- Verify that Secured_Status is UNSECURED when Door_Status, is anything other than CLOSED or UNUSED
13. REPEAT X = (OPEN, UNKNOWN, DOOR_FAULT) DO {
        MAKE (Door_Status = X)
        WAIT (Internal Processing Fail Time)
        VERIFY Secured_Status = UNSECURED
    }
    REPEAT X = (CLOSED, UNUSED) DO {
        MAKE (Door_Status = X)
        WAIT (Internal Processing Fail Time)
        VERIFY Secured_Status = SECURED
    }

-- Verify that Secured_Status is UNSECURED when In_Alarm bit of Status_Flag is True
14. IF (Alarming is supported) THEN
        IF (Alarm_Values is writable) THEN
            WRITE Alarm_Values = { AV: any valid value}
        MAKE (trigger an alarm by using a physical door/lock to create the door alarm state AV)
        WAIT (Internal Processing Fail Time + Time_Delay)
        VERIFY Status_Flags = (TRUE, FALSE, ?, ?)
        VERIFY Secured_Status = UNSECURED
```

7.3.2.X55.1.X6 Door_Unlock_Delay_Time Test

Purpose: To verify that when the Door_Unlock_Delay_Time property has a non-zero value, the output is delayed in unlocking when a PULSE_UNLOCK or EXTENDED_PULSE_UNLOCK is written to the Present_Value and not when UNLOCK is written.

Test Concept: When unlocking the door by writing PULSE_UNLOCK to the Present_Value of the Access Door object, it is verified that the door is still locked for the specified Door_Pulse_Time then the door is unlocked. The same test is done for EXTENDED_PULSE_UNLOCK, but this time it is verified that the door is still locked for the specified Door_Extended_Pulse_Time then the door is unlocked.

Configuration Requirements: The IUT shall be configured with a door control output that can be observed during the test. The Relinquish_Default shall have the value LOCK. All writes to the Present_Value shall be performed at a priority higher than any internal algorithms writing to this property. Door_Unlock_Delay_Time shall be set to a non-zero value which is sufficient to observe the delay and check the status of the lock. Out_Of_Service shall be set to FALSE. Prior to the test the Present_Value shall have the value LOCK and the IUT is in a state that would cause the door to be locked.

Test Steps:

```
-- Test PULSE_UNLOCK
1.  WRITE Present_Value = PULSE_UNLOCK
2.  WAIT (Internal Processing Fail Time)
3.  BEFORE Door_Unlock_Delay_Time
        IF (Lock_Status is present) THEN
            VERIFY Lock_Status = LOCKED
        CHECK (that the door control output is in a state that would cause the door to be locked)

4.  IF (Lock_Status is present) THEN
        VERIFY Lock_Status = UNLOCKED
5.  CHECK (that the door control output is in a state that would cause the door to be unlocked)
6.  WAIT (Door_Pulse_Time)
7.  VERIFY Present_Value = LOCK
8.  IF (Lock_Status is present) THEN
        VERIFY Lock_Status = LOCKED
9.  CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test EXTENDED_PULSE_UNLOCK
10. WRITE Present_Value = EXTENDED_PULSE_UNLOCK
11. WAIT (Internal Processing Fail Time)
12. BEFORE Door_Unlock_Delay_Time
        IF (Lock_Status is present) THEN
            VERIFY Lock_Status = LOCKED
        CHECK (that the door control output is in a state that would cause the door to be locked)

13. IF (Lock_Status is present) THEN
        VERIFY Lock_Status = UNLOCKED
14. CHECK (that the door control output is in a state that would cause the door to be unlocked)
15. WAIT (Door_Extended_Pulse_Time)
16. VERIFY Present_Value = LOCK
17. IF (Lock_Status is present) THEN
        VERIFY Lock_Status = LOCKED
18. CHECK (that the door control output is in a state that would cause the door to be locked)

-- Test UNLOCK
19. WRITE Present_Value = UNLOCK
20. WAIT (Internal Processing Fail Time)
```

```
21. IF (Lock_Status is present) THEN
        VERIFY Lock_Status = UNLOCKED
22. CHECK (that the door control output is in a state that would cause the door to be locked)
```

7.3.2.X55.1.X7 Masked_Alarm_Values Tests

Purpose: To verify that the Masked_Alarm_Values prevents an intrinsic alarm from occurring.

Test Concept: The Access Door is verified to be in an Out_Of_Service stateand is not in an alarm state. Then a non-NORMAL enumeration value of BACnetDoorAlarmState X is written to the Door_Alarm_State and the Access Door object transitions to an alarm state. X is written to the Masked_Alarm_Value and Door_Alarm_State is checked to verify it returned to NORMAL. The sequence is repeated for all non-NORMAL enumeration values of BACnetDoorAlarmState.

Configuration Requirements: The Masked_Alarm_Values list shall be empty at the start of this test. Out_Of_Service shall be set to TRUE to allow writing to the Door_Alarm_State property. If Out_Of_Service is not writeable and cannot be set to TRUE by any other means, this test shall be skipped. The enumeration BACnetDoorAlarmState value X to be used in the test has to be present in either the Alarm_Values or Fault_Values property.

Test Steps:

```
1. VERIFY Status_Flags = (FALSE ?, ?, TRUE)
2. VERIFY Door_Alarm_State = NORMAL
3. REPEAT X = (all valid values of the enumeration BACnetDoorAlarmState except NORMAL)
       DO {
           WRITE Door_Alarm_State = X
           WAIT (Internal Processing Fail Time)
           VERIFY Status_Flags = (TRUE ?, ?, TRUE)
           WRITE Masked_Alarm_Values= { X }
           WAIT (Internal Processing Fail Time)
           VERIFY Door_Alarm_State = NORMAL
           VERIFY Status_Flags = (FALSE ?, ?, TRUE)
           WRITE Masked_Alarm_Values= { }
           WAIT (Internal Processing Fail Time)
       }
```

7.3.2.X55.1.X8 Door_Open_Too_Long Test

Purpose: To verify that the DOOR_OPEN_TOO_LONG condition is generated when the Access Door object is commanded to the LOCK state but the physical door remains open beyond Door_Open_Too_Long_Time.

Test Concept: Setup the Access Door object to trigger alarm on DOOR_OPEN_TOO_LONG state using Alarm_Values and Masked_Alarm_Values. Next, set the physical door to the closed state to confirm that the Access Door object is in NORMAL state. Then, unlock the physical door and set the physical door to the open state. Finally, command the Access Door object to LOCK and verify that the Door_Alarm_State changes to DOOR_OPEN_TOO_LONG after the specified Time_Delay.

Configuration Requirements: This test shall be skipped if the IUT does not support intrinsic alarming. The IUT shall be configured such that it can determine and change the open/closed state of a door. All writes to the Present_Value are at a priority higher than any internal algorithms writing to this property. The Door_Alarm_State shall have the value NORMAL at the start of the test. The Access Door object is configured with DOOR_OPEN_TOO_LONG in the Alarm_Values property and excluded from Masked_Alarm_Values property if present.

Test Steps:

```
1. MAKE (set the physical door to the closed state)
2. VERIFY Door_Alarm_State = NORMAL
3. WRITE Present_Value = UNLOCK
4. MAKE (set the physical door to the open state)
5. WRITE Present_Value = LOCK
6. WAIT (Internal Processing Fail Time)
7. WHILE (Door_Open_Too_Long_Time has not expired) DO {
       VERIFY Door_Alarm_State = NORMAL
   }
```

```
    WAIT (Time_Delay)
8.  VERIFY Door_Alarm_State = DOOR_OPEN_TOO_LONG
```

BTL-15.2m-3: COV-B for Access Door [BTLWG-408]

Overview:

Access Door object type support COV reporting, and which specify parameters in the COVNotification that are unique to the object type. This proposal specifies an approach for the support of testing those which is similar as is used in the Loop object type, which similarly has parameters in the COVNotification that are unique to the object type.

Changes:

[In BTL Checklist, a section each with "Contact BTL for interim tests…" already exists, under Data Sharing - Change of Value - A, for the Access Door object type. That is positioned before "Can Subscribe for COV from Lighting Output/Binary Lighting Output objects" sections.]

[In BTL Checklist, a section each with "Contact BTL for interim tests…" already exists, under Data Sharing - Change of Value – B, for the Access Door object type. That is positioned before "Can Subscribe for COV from Lighting Output/Binary Lighting Output objects" sections.]

[In BTL Test Plan, modify section in 4.9 and 4.10 for the Access Door object type]

4.9 Data Sharing - Change Of Value - A

4.9.19 Can Subscribe for COV from Access Door Objects

The IUT supports change of value notifications for at least one object of type Access Door.

~~**135.1-2013 - 9.2.1.1 - Change of Value Notifications**~~		
	~~**Test Conditionality**~~	~~Must be executed.~~
	~~**Test Directives**~~	~~Test one instance of each object type where Present_Value is of data type Access Door.~~
	~~**Testing Hints**~~	
~~**135.1-2013 - 9.3.2 - Change of Value Notifications**~~		
	~~**Test Conditionality**~~	~~Must be executed.~~
	~~**Test Directives**~~	~~Test one instance of each object type where Present_Value is of data type Access Door.~~
	~~**Testing Hints**~~	
BTL - 9.2.1.X5 - ConfirmedCOVNotification from Access Door		
	Test Conditionality	*Either 9.2.1.X5 or 9.3.1.X6 must be executed*
	Test Directives	*Test at least one instance where object type is an Access Door.*
	Testing Hints	
BTL - 9.3.1.X6 - UnconfirmedCOVNotification from Access Door		
	Test Conditionality	*Either 9.2.1.X5 or 9.3.1.X6 must be executed*
	Test Directives	*Test at least one instance where object type is an Access Door.*
	Testing Hints	

4.10 Data Sharing - Change Of Value - B

4.10.28 Supports COV for Access Door Objects

The IUT supports change of value notifications for at least one object of type Access Door.

~~Contact BTL for interim tests for this object.~~

BTL - 8.2.X11 - Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property		
	Test Conditionality	*This may be skipped if **BTL - 8.3.X14** is executed against an Access Door object.*
	Test Directives	*Test at least one instance where object type is an Access Door.*
	Testing Hints	

BTL - 8.3.X14 - Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property		
	Test Conditionality	*This may be skipped if **BTL - 8.2.X11** is executed against an Access Door object.*
	Test Directives	*Test at least one instance where object type is an Access Door.*
	Testing Hints	

[In BTL Specified Tests, add new test in existing section 8.2 for the Access Door.]

8.2 ConfirmedCOVNotification Service Initiation Tests

8.2.X11 Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property

Purpose: To verify that the IUT can initiate ConfirmedCOVNotification service requests conveying a change of the Present_Value property of Access Door objects.

Test Concept: A subscription for COV notifications is established, using a Lifetime of L. L shall be set to a value less than 24 hours and large enough to complete the test. The Present_Value of the monitored object is changed, and a notification shall be received. The Present_Value may be changed using the WriteProperty service or by another means. For some implementations it may be necessary to write to the Out_Of_Service property first to accomplish this task. For implementations where it is not possible to write to these properties at all the vendor shall provide an alternative trigger mechanism to accomplish this task. All these methods are equally acceptable.

Configuration Requirements: At the beginning of the test, the Out_Of_Service property shall have a value of FALSE. Select an object where Present_Value is not expected to change outside the tester's control, or which has a writable Out_Of_Service. If no object has a Door_Alarm_State property, then step 9,10,11 shall be skipped. For implementations where it is not possible to write Out_Of_Service or change the Status_Flags by any other means, step 5,6,7 shall be skipped

Test Steps:

```
REPEAT X = (one supported object of type Access Door) DO {
1.  TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =    (any value > 0chosen by the TD),
        'Monitored Object Identifier' =      X,
        'Issue Confirmed Notifications' =    TRUE,
        'Lifetime' =                         L
2.  RECEIVE BACnet-SimpleACK-PDU
3.  BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =       (the same value used in step 1),
            'Initiating Device Identifier' =  IUT,
            'Monitored Object Identifier' =  X,
            'Time Remaining' =               (any value appropriate for the Lifetime selected),
            'List of Values' =               (the initial Present_Value, initial Status_Flags, and
                                             Door_Alarm_State if X has a Door_Alarm_State property)
4.  TRANSMIT BACnet-SimpleACK-PDU
5.  IF (Out_Of_Service is writable) THEN
        WRITE Out_Of_Service = TRUE
    ELSE
        MAKE (Status_Flags = any value that differs from initial Status_Flags)

6.  BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =       (the same value used in step 1),
            'Initiating Device Identifier' =        IUT,
            'Monitored Object Identifier' =         X,
            'Time Remaining' =                      (any value appropriate for the Lifetime selected),
            'List of Values' =                      (ReportedPV=current Present_Value, new Status_Flags, and
                                                  Door_Alarm_State if X has a Door_Alarm_State property)
7.  TRANSMIT BACnet-SimpleACK-PDU
8.  IF (Present_Value is writable) THEN
        WRITE X,Present_Value = (any value that differs from ReportedPV)
    ELSE
        MAKE (Present_Value = any value that differs from ReportedPV)
9.  BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =       (the same value used in step 1),
            'Initiating Device Identifier' =  IUT,
            'Monitored Object Identifier' = X,
            'Time Remaining' =    (any value appropriate for the Lifetime selected),
            'List of Values' =    (the new Present_Value, new Status_Flags, and Door_Alarm_State if X has a
                                  Door_Alarm_State property)
10. TRANSMIT BACnet-SimpleACK-PDU
11. IF (Door_Alarm_State is now writable) THEN
        WRITE Door_Alarm_State = (any value that differs from its initial Door_Alarm_State)
    ELSE
        MAKE (Door_Alarm_State = any value that differs from its initial Door_Alarm_State)
12. BEFORE Notification Fail Time
        RECEIVE ConfirmedCOVNotification-Request,
            'Subscriber Process Identifier' =       (the same value used in Step 1),
            'Initiating Device Identifier' =  IUT,
            'Monitored Object Identifier' = X,
            'Time Remaining' =    (any value appropriate for the Lifetime selected),
            'List of Values' =    (the new Present_Value, new Status_Flags, and Door_Alarm_State)
13. TRANSMIT BACnet-SimpleACK-PDU

14. TRANSMIT SubscribeCOV-Request,
        'Subscriber Process Identifier' =    (the same value used in the SubscribeCOV-Request),
        'Monitored Object Identifier' =      X
15. RECEIVE BACnet-SimpleACK-PDU
16. IF (Out_Of_Service is writable) THEN
        WRITE Out_Of_Service =               FALSE
```

[In BTL Specified Tests, Add new test for Access Door Object to section 8.3]

8.3 UnconfirmedCOVNotification Service Initiation Tests

8.3.X14 Change of Value Notification from an Access Door object Present_Value, Status_Flags and Door_Alarm_State property

Purpose: To verify that the IUT can initiate UnconfirmedCOVNotification service requests conveying a change of the Present_Value, Status_Flag and Door_Alarm_State property of Access Door objects.

Test Steps: The steps for this test case are identical to the test steps in 8.2.X11 except that the SubscribeCOV service request in step 1 shall have a value of FALSE for the 'Issue Confirmed Notifications' parameter, all the ConfirmedCOVNotification requests shall be UnconfirmedCOVNotification requests, and there is no acknowledgment of the unconfirmed services.

[In BTL Specified Tests, add new tests for Access Door in section 9.2.1]

9.2.1 Positive ConfirmedCOVNotification Service Execution Tests

9.2.1.X5 ConfirmedCOVNotification from Access Door Object

Purpose: To verify that the IUT can execute ConfirmedCOVNotification requests from Access Door objects.

Test Steps:

```
1.  RECEIVE SubscribeCOV-Request,
        'Subscriber Process Identifier' =    (any valid process identifier value > 0),
```

'Monitored Object Identifier' = (any Access Door object, X),
'Issue Confirmed Notifications ' = TRUE,
'Lifetime' = (a value greater than one minute)

2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT ConfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the process identifier used in step 1),
   'Initiating Device Identifier' = TD,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (the time remaining in the subscription),
   'List of Values' = (the initial Present_Value, initial Status_Flags, and Door_Alarm_State if X has a Door_Alarm_State property)
4. RECEIVE BACnet-SimpleACK-PDU
5. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)

[In BTL Specified Tests, add new test for Access Door for Unconfirmed COV Notification]

9.3.1.X6 UnconfirmedCOVNotification from Access Door Object

Purpose: To verify that the IUT can execute UnconfirmedCOVNotification requests from Access Door objects.

Test Steps:

1. RECEIVE SubscribeCOV-Request,
   'Subscriber Process Identifier' = (any valid process identifier value > 0),
   'Monitored Object Identifier' = (any Access Door object, X),
   'Issue Confirmed Notifications ' = FALSE,
   'Lifetime' = (a value greater than one minute)
2. TRANSMIT BACnet-SimpleACK-PDU
3. TRANSMIT UnconfirmedCOVNotification-Request,
   'Subscriber Process Identifier' = (the process identifier used in step 1),
   'Initiating Device Identifier' = TD,
   'Monitored Object Identifier' = X,
   'Time Remaining' = (the time remaining in the subscription),
   'List of Values' = (the initial Present_Value, initial Status_Flags, and Door_Alarm_State if X has a Door_Alarm_State property)
4. CHECK (to ensure that any appropriate functions defined by the manufacturer, such as displaying information on a workstation screen are carried out)