

## Clarification Request

**References:** BTL Specified Tests 12.0.final tests 9.22.2.3, 9.23.2.6, 9.14.2.2, 9.16.2.5

**Date of BTL-WG Response:** Dec 12, 2012

### Background:

Each of these tests involves a TRANSMIT which would never be observed from an implementation coded by someone competent in their understanding of BACnet. The Result(-) outcome can entirely be foreseen. It is difficult to justify requiring each IUT to detect and differentiate an unacceptable datatype from an unacceptable value. Especially since the Initiating implementation already cannot be simply looking for a single error code, as it must also expect BACnet-Reject-PDU with either of two possible Reject-Reason codes. Requiring each IUT to add code for something that would never be observed from a correct implementation is at-odds with over-arching philosophy of the BTL Test Plan in that regard.

### From BTL Specified Tests 12.0

#### 9.14.2.2 Adding a List Element With an Invalid Datatype

Reason for change: Success criteria should specify 'First Failed Element' = 1 and the additional error conditions are now accepted.

Purpose: To verify the ability of the IUT to correctly respond to an AddListElement service request to add an element with an invalid datatype to a list.

#### Test Steps:

1. TRANSMIT AddListElement-Request,
  - 'Object Identifier' = L,
  - 'Property Identifier' = ListProp,
  - 'List of Elements' = (a single element with a datatype inappropriate for this property)
2. RECEIVE AddListElement-Error,
  - Error Class = PROPERTY,
  - Error Code = INVALID\_DATATYPE,
  - 'First Failed Element' = 01 /
  - (BACnet-Reject-PDU
  - Reject Reason = INVALID\_PARAMETER\_DATATYPE) /
  - (BACnet-Reject-PDU
  - Reject Reason = INVALID\_TAG)

Notes to Tester: value selected for step 1 is 'inappropriate', not a value which is 'allowed' but not supported by this instance of the property. I.e. it is not one of the datatypes that would ever be supported by an instance of this property in this object type. DATATYPE\_NOT\_SUPPORTED is only correct when the

datatype requested *is* supported, for example in a CHOICE, by this property in this object type, but not supported by this instance of the property.

### 9.16.2.5 Attempting to Create an Object with an Object Identifier Object Specifier and an Error in the Initial Values

Reason for Change: INTERPRETATION IC 135-2004-28

Purpose: To verify the correct execution of the CreateObject service request when an object identifier is used as the object specifier and a list of initial property values containing an invalid value is provided.

Test Steps:

1. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (any unique object identifier of a type that is creatable *and an instance number that is creatable*)  
     'List Of Initial Values' = (a list of ~~two~~*one* or more properties and their initial values, *that the IUT will accept initial values for*, with one of the values being out of range)
2. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  4) THEN  
     RECEIVE CreateObject-Error-PDU,  
         Error Class = PROPERTY,  
         Error Code = VALUE\_OUT\_OF\_RANGE  
         'First Failed Element Number' = (the position in the 'List Of Initial Values' with the invalid value)  
     ELSE  
         RECEIVE CreateObject-Error,  
             Error Class = PROPERTY,  
             Error Code = VALUE\_OUT\_OF\_RANGE  
             INVALID\_DATATYPE OTHER  
             'First Failed Element Number' = (the position in the 'List Of Initial Values' with the invalid value)  
     3. CHECK(Verify that the new object was not created)
4. TRANSMIT CreateObject-Request,  
     'Object Specifier' = (~~any unique~~ object identifier from step 1 ~~of a type that is creatable~~),  
     'List Of Initial Values' = (a list of ~~two~~*one* or more properties and their initial values, *IUT will accept initial values for*, with one of the values being an inappropriate datatype)
5. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  4) THEN  
     RECEIVE  
         CreateObject-Error-PDU,  
             Error Class = PROPERTY,  
             Error Code = INVALID\_DATATYPE  
             'First Failed Element Number' = (the position in the 'List Of Initial Values' with the invalid value) /  
             (BACnet-Reject-PDU  
                 Reject Reason = INVALID\_PARAMETER\_DATATYPE) /  
             (BACnet-Reject-PDU  
                 Reject Reason = INVALID\_TAG)  
     ELSE  
         RECEIVE  
             CreateObject-Error,  
                 Error Class = PROPERTY,  
                 Error Code = VALUE\_OUT\_OF\_RANGE  
                 INVALID\_DATATYPE / OTHER

'First Failed Element Number' = (the position in the 'List Of Initial Values' with the *invalid* value) |  
 (BACnet-Reject-PDU  
 Reject Reason = INVALID\_PARAMETER\_DATATYPE /  
 INVALID\_TAG)  
 6. TRANSMIT ReadProperty-Request,  
 'Object Identifier' = (the 'Object Identifier' used in step 1),  
 'Property Identifier' = ~~(any required property of the specified object)~~ *Object\_Name*  
 7. IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  4) THEN  
 RECEIVE BACnet-Error-PDU,  
 Error Class = OBJECT,  
 Error Code = UNKNOWN\_OBJECT  
 ELSE  
 RECEIVE BACnet-Error-PDU  
 Error Class = OBJECT,  
 Error Code = UNKNOWN\_OBJECT | NO\_OBJECTS\_OF\_SPECIFIED\_TYPE | OTHER

### 9.22.2.3 Writing with a Property Value Having the Wrong Datatype

Reason for Change: Relax Tests of INVALID\_DATATYPE per INTERPRETATION IC 135-2004-28.  
 Modified Test to remove dependency on EPICS values.

Purpose: To verify that the IUT correctly responds to an attempt to write a property value that has an invalid datatype.

Test Concept: The TD shall select an object in the IUT that contains a writable property designated P1. An attempt will be made to write to this property using an invalid datatype. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ X = (Object1), P1  
~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~  
 2. TRANSMIT WriteProperty-Request,  
 'Object Identifier' = Object1,  
 'Property Identifier' = P1,  
 'Property Value' = (any value with an invalid datatype)  
 3. ~~IF (Protocol\_Revision is present and Protocol\_Revision  $\geq$  4) THEN~~  
~~RECEIVE BACnet-Error-PDU,~~  
~~Error Class = PROPERTY,~~  
~~Error Code = INVALID\_DATATYPE~~  
~~ELSE~~  
 RECEIVE  
 (BACnet-Error PDU,  
 Error Class = PROPERTY,  
 Error Code = INVALID\_DATATYPE) |  
 (BACnet-Reject-PDU  
 Reject Reason = INVALID\_PARAMETER\_DATATYPE) |  
 (BACnet-Reject-PDU  
 Reject Reason = INVALID\_TAG)  
 4. VERIFY (Object1), P1 = X~~(the value defined for this property in the EPICS)~~

### 9.23.2.6 Writing with a Property Value Having the Wrong Datatype

Reason for Change: Modified test to remove dependency on EPICS values.

Purpose: This test case verifies that the IUT correctly responds to an attempt to write a property value that has an invalid datatype. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, in the IUT that contains a writable property designated P1.

An attempt will be made to write to this property using an invalid datatype. If no object supports writable properties, then this test shall be omitted.

Test Steps:

1. READ X = (Object1), P1
- ~~1. VERIFY (Object1), P1 = (the value defined for this property in the EPICS)~~
2. TRANSMIT WritePropertyMultiple-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any value with an invalid datatype)
3. RECEIVE WritePropertyMultiple-Error,
  - 'Error Class' = PROPERTY,
  - 'Error Code' = INVALID\_DATATYPE,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1
4. VERIFY (Object1), P1 = X~~(the value defined for this property in the EPICS)~~

Note also that 135.1-2011 gives indication to its readers, in two tests **9.22.2.5** and **9.23.2.8** which are similar to these, but which the BTL Test Plan does not reference, though they nevertheless influence implementers who read 135.1-2011, that show ambivalence about which particular error code is observed.

## From 135.1-2011

### 9.22.2.5 Writing To Non-Existent Objects

Purpose: This test case verifies that the IUT can execute WriteProperty service requests when the object specified in the service request does not exist. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, that does not exist in the IUT. Object1 shall be of a type supported by the IUT. An attempt will be made to write to a property, designated P1, in this non-existent object. P1 shall refer to a standard property that is supported by this object type in the IUT.

Test Steps:

1. TRANSMIT WriteProperty-Request,
  - 'Object Identifier' = Object1,
  - 'Property Identifier' = P1,
  - 'Property Value' = (any value of the correct datatype for this property)
2. RECEIVE BACnet-Error PDU,
  - Error Class = OBJECT,
  - Error Code = UNKNOWN\_OBJECT

Passing Result: While OBJECT::UNKNOWN\_OBJECT is the desired error for this condition, in some implementations other error conditions may be checked before the existence of the object itself. The other errors that are acceptable are:

- PROPERTY::UNKNOWN\_PROPERTY,
- PROPERTY::WRITE\_ACCESS\_DENIED,
- PROPERTY::INVALID\_DATATYPE,

PROPERTY::VALUE\_OUT\_OF\_RANGE and  
 RESOURCES::NO\_SPACE\_TO\_WRITE\_PROPERTY.

### 9.23.2.8 Writing To Non-Existent Objects

Purpose: This test case verifies that the IUT can execute WritePropertyMultiple service requests when the object specified in the service request does not exist. This test shall only be performed if Protocol\_Revision is present and has a value greater than or equal to 4.

Test Concept: The TD shall select an object, designated Object1, that does not exist in the IUT. Object1 shall be of a object type supported by IUT. An attempt will be made to write to a property, designated P1, in this non-existent object. P1 shall refer to a standard property that is supported by this object type in the IUT.

Test Steps:

1. TRANSMIT WritePropertyMultiple-Request,  
     'Object Identifier' =       Object1,  
     'Property Identifier' =     P1,  
     'Property Value' =         (any value of the correct datatype for this property)
2. RECEIVE WritePropertyMultiple-Error,  
     Error Class =            OBJECT,  
     Error Code =            UNKNOWN\_OBJECT,  
     objectIdentifier =     Object1,  
     propertyIdentifier =   P1

Passing Result: While OBJECT::UNKNOWN\_OBJECT is the desired error for this condition, in some implementations, other error conditions may be checked before the existence of the object itself. The other errors that are acceptable are:

PROPERTY::UNKNOWN\_PROPERTY,  
 PROPERTY::WRITE\_ACCESS\_DENIED,  
 PROPERTY::INVALID\_DATATYPE,  
 PROPERTY::VALUE\_OUT\_OF\_RANGE and  
 RESOURCES::NO\_SPACE\_TO\_WRITE\_PROPERTY.

#### Question:

Should the VALUE\_OUT\_OF\_RANGE error code response also be acceptable if the IUT receives any of these requests?

#### Response:

No. It makes a user visible difference because INVALID\_DATATYPE points out an error in implementation, while VALUE\_OUT\_OF\_RANGE points out an error in user choice. They are useful to differentiate.