

Clarification Request

References: Add-BTL Test Package 12.0-g-v3.doc

BTL Specified Tests change 13.8.1.1

Date of BTL-WG Response: original response 2014-03-06
Updated response 2021-12-09

Background:

13.8.1.1 Execution of Full Backup and Restore Procedure

Purpose: This test case verifies that the IUT can execute a full Backup and Restore procedure.

Test Concept: This test takes the IUT through a successful Backup and then a successful Restore procedure. The Database_Revision and Last_Restore_Time properties are noted before the procedure begins for later comparison. The IUT is then commanded to enter the Backup state; all the files are read, and the IUT is commanded to end the backup. If the Database_Revision property can be changed by means other than the restore procedure, it is modified and checked to ensure that it incremented correctly; then the IUT is commanded to enter the Restore state. If the file objects do not exist on the IUT, the TD will create them in the IUT. The files are then truncated to size 0, the file contents are written to the IUT, and the IUT is commanded to end the restore. The Database_Revision and Last_Restore_Time properties are checked to ensure that they incremented or advanced correctly.

For IUTs that use Stream Access when performing the AtomicReadFile and AtomicWriteFile services, a Maximum Requested Octet Count (MROC) and a Maximum Write Data Length (MWDL) shall be calculated before starting the test. These values shall be used during the test. MROC shall be 16 less than the minimum of the TD's Max_APDU_Length_Accepted and the IUT's maximum transmittable APDU length. MWDL shall be 21 less than the minimum of the TD's maximum transmittable APDU length and the IUT's Max_APDU_Length_Accepted.

Test Steps:

1. READ DR1 = Database_Revision
2. READ LRT1 = Last_Restore_Time
3. READ OL1 = Object_List
4. REPEAT X = (1 through length of OL1) DO {
 READ NAMES[X] = (OL1[X]), Object_Name
}
5. IF (Protocol_Revision is present and Protocol_Revision \geq 10) THEN
 READ BPT = Backup_Preparation_Time

```

    READ RPT = Restore_Preparation_Time
    READ RCT = Restore_Completion_Time
    VERIFY Backup_And_Restore_State = IDLE
6. TRANSMIT ReinitializeDevice-Request,
    'Reinitialized State of Device' = STARTBACKUP,
    'Password' = (any valid password)
7. RECEIVE BACnet-Simple-ACK-PDU
8. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT BPT
    READ BRSTATE = Backup_And_Restore_State
    READ CF = Configuration_Files
    WHILE (BRSTATE = PREPARING_FOR_BACKUP) DO {
        WAIT 1 second
        READ BRSTATE = Backup_And_Restore_State
        IF CF is an empty list THEN
            READ CF = Configuration_Files
        IF CF is a non-empty list THEN
            READ X = (the file referenced by Configuration_Files[1]).Name
        }
        CHECK (BRSTATE = PERFORMING_A_BACKUP)
9. READ CF = Configuration_Files
10. CHECK (CF is a non-empty array of BACnetObjectIdentifiers referring to File
objects)
11. REPEAT X = (each entry in CF) DO {
    READ Y = X, File_Access_Method
    IF (Y = RECORD_ACCESS)
        WHILE (the last read resulted in an Ack with 'End Of File' == FALSE) DO {
            TRANSMIT AtomicReadFile-Request,
                'Object Identifier' = X,
                'File Start Record' = (the next unread record),
                'Requested Record Count' = 1
            RECEIVE AtomicReadFile-ACK,
                'End Of File' = TRUE | FALSE,
                'File Start Record' = Z,
                'Requested Record Count' = 1
                'Returned Data' = (File contents)
            | Error-PDU -- only acceptable for the first record and only when
there are no records in the file
                'Error Class' = SERVICES,
                'Error Code' =
INVALID_FILE_START_POSITION
        }
    ELSE
        WHILE (the last read did not indicate 'End Of File') DO {
            TRANSMIT AtomicReadFile-Request,
                'Object Identifier' = X,

```

```

        'File Start Position' = (the next unread octet),
        'Requested Octet Count' = MROC
    RECEIVE AtomicReadFile-ACK,
        'End Of File' = TRUE | FALSE,
        'File Start Position' = (the next unread octet)
        'File Data' = (File contents of length MROC if 'End Of
File' is FALSE
                                or of length MROC or less if 'End Of File' is
TRUE)
        | Error-PDU -- only acceptable for the first record and only when
there are no records in the file
        'Error Class' = SERVICES,
        'Error Code' =
INVALID_FILE_START_POSITION
    }
}
12. TRANSMIT ReinitializeDevice-Request,
    'Reinitialize State Of Device' = ENDBACKUP,
    'Password' = (any valid password)
13. RECEIVE BACnet-Simple-ACK-PDU
14. VERIFY System_Status != BACKUP_IN_PROGRESS
15. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    VERIFY Backup_And_Restore_State = IDLE
16. IF (Database_Revision is changeable) THEN
    MAKE (the configuration in the IUT different, such that the Database_Revision
property increments)
    VERIFY Database_Revision <> DR1
    READ DR2 = Database_Revision
    CHECK (DR1 <> DR2)
17. TRANSMIT ReinitializeDevice-Request,
    'Reinitialize State Of Device' = STARTRESTORE,
    'Password' = (any valid password)
18. RECEIVE BACnet-Simple-ACK-PDU
19. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RPT
    READ BRSTATE = Backup_And_Restore_State
    WHILE (BRSTATE = PREPARING_FOR_RESTORE) DO {
        WAIT 1 second
        READ BRSTATE = Backup_And_Restore_State
    }
    CHECK (BRSTATE = PERFORMING_A_RESTORE)
20. READ OL2 = Object_List
21. REPEAT X = (entry in CF) DO {
    IF (X is not in OL2)
        TRANSMIT CreateObject-Request
        'Object Identifier' = X

```

```

    RECEIVE CreateObject-ACK
        'Object Identifier' = X
    READ FS = X, File_Size
    IF (File_Size is not equal to the size of the backed up file)
        WRITE X, File_Size = 0
    IF (Y = RECORD_ACCESS)
        READ RC = X, Record_Count
        IF (RC is not equal to the number of records of the backed up file)
            WRITE X, Record_Count = 0
        TRANSMIT AtomicWriteFile-Request
            'File Identifier' = X
            'File Start Record' = 0
            'Record Data' = (file content for first record obtained in step 11)
        RECEIVE AtomicWriteFile-ACK
            'File Start Record' = 0
        REPEAT REC = (each record in the backup of this file) {
            TRANSMIT AtomicWriteFile-Request
                'File Identifier' = X
                'File Start Record' = -1
                'Record Count' = 1
                'Record Data' = REC
            RECEIVE AtomicWriteFile-ACK
                'File Start Record' = (the record number)
        }
    ELSE
        READ FS = X, File_Size
        IF (File_Size is not equal to the size of the backed up file)
            WRITE X, File_Size = 0
        REPEAT Z = (0 through the file size, in increments of MWDL) DO {
            TRANSMIT AtomicWriteFile-Request
                'File Identifier' = X
                'File Start Position' = Z
                'Record Data' = (file contents obtained from the backup,
the number of octets
being the lesser of (file size - Z) and
MWDL)
            RECEIVE AtomicWriteFile-ACK
                'File Start Position' = Z
        }
    }
22. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    VERIFY Backup_And_Restore_State = PERFORMING_A_RESTORE
23. TRANSMIT ReinitializeDevice-Request,
    'Reinitialize State Of Device' = ENDRESTORE,
    'Password' = (any valid password)
24. RECEIVE BACnet-Simple-ACK-PDU

```

```

25. IF (Protocol_Revision is present and Protocol_Revision ≥ 10) THEN
    WAIT RCT
    VERIFY Backup_And_Restore_State = IDLE
26. READ DR3 = Database_Revision
27. CHECK (DR3 <> DR1)
28. IF (Database_Revision was changed in step 16) THEN
    CHECK (DR3 <> DR2)
29. VERIFY Last_Restore_Time > LRT1
30. READ OL3 = Object_List
31. CHECK (that OL1 and OL3 contain the same set of objects)
32. REPEAT X = (1 through length of OL1) DO {
    VERIFY (OL1[X]), Object_Name = NAMES[X]
}

```

Problem:

in case of a file with record access the restore procedure as written tries to do the size checking and resizing using the property File_Size as if it was stream access (marked in red above). The correct method for files with record access should be to use the Record_Count property in a similar way (added text marked in green).

Questions:

1. Should the test be changed as outlined above?
2. There is a create object request (marked in yellow). Especially in case a device has mixed backup files, some with RECORD_ACCESS, some with STREAM_ACCESS it might be necessary if the test employs initial parameters specifying the access method. Should the test be allowed to use additional parameters?

Original Response:

"The suggested revision, when in RECORD_ACCESS to utilize Record_Count rather than File_Size, is a beneficial change. It will be incorporated into a revised test in the Test Plan.

The CreateObject request shall not provide initial values. The IUT has to be able to accept the CreateObject request with no initial values, and itself produce the File with the needed File_Access_Method. The Object_Identifier is provided in each CreateObject request and that shall be enough information."

Updated Response:

Subsequent to the BTL-WG's original ruling, the SSPC modified the standard to clarify that both record and stream based files are to be cleared by writing to File_Size. This change in the standard contradicts the original CR response for question 1, and as such the proposed changes will not be applied to the test.