|  |  |  |
| --- | --- | --- |
| Unit Test plan document for Call converter | February 16  2011 | |
| Call converter module converts set of TS packets to Juniper’s proprietary media fundamental unit. This involves complex parsing for video frame start and ADTS frame start. This document talks about the unit test plan for the same module. | | Ver 0.0 |

Table of Contents

[Unit test plan for Call Converter (TS to MFU) 4](#_Toc285633679)

[1. Call converter API 4](#_Toc285633680)

[2. Input and output 4](#_Toc285633681)

[2.1. Inputs 4](#_Toc285633682)

[2.2. Outputs 4](#_Toc285633683)

[3. Data structures 4](#_Toc285633684)

[3.1. Ts2mfu context data structure 4](#_Toc285633685)

[3.2. MFU Header 5](#_Toc285633686)

[4. Call converter unit test wrapper and its Input and output 5](#_Toc285633687)

[5. Unit test plan 6](#_Toc285633688)

[5.1. Failure cases 6](#_Toc285633689)

[5.1.1. Missing sync byte 6](#_Toc285633690)

[5.1.2. Missing continuity counter 6](#_Toc285633691)

[5.1.3. Missing payload start indicator 6](#_Toc285633692)

[5.2. Functionality check 6](#_Toc285633693)

[5.2.1. Video packets functionality 6](#_Toc285633694)

[5.2.2. Audio packets functionality 6](#_Toc285633695)

[5.2.3. Audio frame packing test 7](#_Toc285633696)

[5.2.3.1. Packing with start offset as zero 7](#_Toc285633697)

[5.2.3.2. Packing with start offset as 10 bytes 7](#_Toc285633698)

[5.2.3.3. Packing with start offset - variable 7](#_Toc285633699)

[5.2.3.4. Packing 2 frames in a TS packet 7](#_Toc285633700)

[5.2.3.5. Packing 3 frames in a TS packet 7](#_Toc285633701)

# Unit test plan for Call Converter (TS to MFU)

## Call converter API

mfu\_data\_t\* //output – MFU data structure  
**mfp\_ts2mfu**(  
uint32\_t\* mstr\_pkts, //Input – array of video packets address offset for a chunk  
uint32\_t mstr\_npkts, //Input – number of video packets in input  
uint32\_t\* slve\_pkts, // Input – array of audio packets address for a chunk  
uint32\_t slve\_npkts, // Input – number of audio packets in input  
uint8\_t \*data, //Input – the input buffer on which input video and audio //packets exist  
mfub\_t \*mfubox\_header, //Output – mfub header  
ts2mfu\_cont\_t \*ts2mfu\_cont ) //Input + output – context data structure

## Input and output

### Inputs

uint32\_t\* mstr\_pkts, //Input – array of video packets address for a chunk  
uint32\_t mstr\_npkts, //Input – number of video packets in input  
uint32\_t\* slve\_pkts, // Input – array of audio packets address for a chunk  
uint32\_t slve\_npkts, // Input – number of audio packets in input  
uint8\_t \*data, //Input – the input buffer on which input video and audio //packets exist  
ts2mfu\_cont\_t \*ts2mfu\_cont ) //Input + output – context data structure

### Outputs

mfu\_data\_t\* //output – MFU data structure  
mfub\_t \*mfubox\_header, //Output – mfub header  
ts2mfu\_cont\_t \*ts2mfu\_cont ) //Input + output – context data structure

## Data structures

### Ts2mfu context data structure

typedef struct ts2mfu\_cont\_tt {  
 uint32\_t is\_audio;  
 uint32\_t num\_tspkts\_retx; //number of audio TS packets to be transmitted

uint32\_t last\_pkt\_start\_offset; //the last TS packet has to be filled with zeros just before the  
 uint32\_t last\_pkt\_end\_offset; //next frame header. Start offset and end offset limits the zero  
 // filling code  
 uint32\_t prev\_chunk\_frame\_end; //last seen audio packets continuity counter in previous chunk  
 uint32\_t curr\_chunk\_frame\_start; //the first audio packet continuity counter in current chunk  
 uint32\_t curr\_chunk\_frame\_end; //the last audio packet continuity counter in current chunk  
 uint32\_t first\_instance; //flag to represent first chunk in the session  
 uint8\_t \*prev\_aud\_chunk; //previous audio chunk  
 uint32\_t prev\_aud\_chunk\_size; //previous audio chunk size  
 uint32\_t prev\_aud\_chunk\_cont[3]; //0 – previous chunk number of packets retransmitted  
 //1 – previous chunk last packet start offset  
 //2 – previous chunk last packet end offset  
 uint32\_t last\_audio\_index; //audio Lookup table index  
} ts2mfu\_cont\_t;

### MFU Header

Please refer to MFU document for detailed description of this data structure. Also see nkn\_vpe\_mfu\_defs.h.

typedef struct tag\_mfub

{

uint16\_t version; /\*\*< version of the MFU box; MajorVersion(8 bytes):MinorVersion(8 bytes)\*/ uint16\_t program\_number;/\*our encaps number\*/  
 uint16\_t stream\_id;/\*unique id to identify each streaam\*/  
 uint16\_t flags;  
 …

uint64\_t offset\_vdat; /\*\*< offset to the first video elementary stream unit \*/  
uint64\_t offset\_adat;/\*\*< offset to the first audio elementary stream unit \*/  
uint32\_t mfu\_size;/\* size of the entire mfu \*/

} mfub\_t;

## Call converter unit test wrapper and its Input and output

The ts2mfu API test wrapper is   
nokeena\src\lib\nvsd\parser\ts2mfu\_test\src\main\_new\_ts2mfu.c. The command line tool takes in three arguments.

Argv[1] is file name of input audio TS packets collection.  
Argv[2] is the file name of input video TS packets collection.  
Argv[3] is the output file name for MFU data dumping.

The shell script to compile the unit test executable is   
nokeena\src\lib\nvsd\parser\ts2mfu\_test\bin\ build\_unit\_test.sh

## Unit test plan

### Failure cases

### Missing sync byte

Create set of video TS packets and audio TS packets for 2 seconds chunks. Remove sync byte (0x47) in few ts packets and check whether ts2mfu is returning NULL mfu\_data\_t.

### Missing continuity counter

Create set of video TS packets and audio TS packets for 2 seconds chunks. Remove 1 random video packet and check the mfu\_data returned. Expected output is NULL buffer. Remove 1 random audio packet and check the mfu\_data returned. Expected output is NULL buffer.

### Missing payload start indicator

Create set of video TS packets and audio TS packets for 2 seconds chunks. In video TS packets list, remove all payload start indicator packets and check whether output returned in NULL. In audio TS packets list, remove all payload start indicator packets and check whether output returned in NULL.

### Functionality check

### Video packets functionality

Create set of video TS packets with known set of video frames. Feed the video.ts to ts2mfu converter and get the mfu output. Check the following values:

Vmd->sample\_count == known frame number value for the file  
SUM(Vmd-> sample\_duration) == input video TS packets last frame PTS – first TS packet PTS  
CHECK\_ALL(vmd->sample\_sizes) == known sample size for the given input video TS packets  
CHECK\_NALU\_START(vmd->mdat\_pos + SUM(vmd->sample\_size,0..n-1)) for the Nth frame. The NALU start code can be 00 00 01 or 00 00 00 01 (with/without emulation)

### Audio packets functionality

Create set of audio TS packets with known set of video frames. Feed the audio.ts to ts2mfu converter and get the mfu output. Check the following values:

amd->sample\_count == known frame number value for the file  
SUM(amd-> sample\_duration) == input audio TS packets last frame PTS – first TS packet PTS  
CHECK\_ALL(amd->sample\_sizes) == known sample size for the given input audio TS packets  
CHECK\_ADTS\_START(amd->mdat\_pos + SUM(amd->sample\_size,0..n-1)) for the Nth frame. ADTS header is can be 0xff followed by 0xf0, 0xf1, 0xf8, or 0xf9.

### Audio frame packing test

### Packing with start offset as zero

Take a set of 60 audio frames (ADTS + AAC LC). Write a function to create TS packets for the set of audio frames with packing i.e, first audio frame is followed by second audio frame. Feed the set of audio TS packets as input to ts2mfu and check the output mfu header for following values:

amd->sample\_count == known frame number value for the file  
SUM(amd-> sample\_duration) == input audio TS packets last frame PTS – first TS packet PTS  
CHECK\_ALL(amd->sample\_sizes) == known sample size for the given input audio TS packets  
CHECK\_ADTS\_START(amd->mdat\_pos + SUM(amd->sample\_size,0..n-1)) for the Nth frame. ADTS header is can be 0xff followed by 0xf0, 0xf1, 0xf8, or 0xf9.

### Packing with start offset as 10 bytes

Repeat the above test with starting offset as 10 bytes. That is the first TS packet adaption field is enabled and filled with 10 0xff bytes. Validating output remains same as before test.

### Packing with start offset - variable

Repeat the above test with starting offset as 20 bytes, 30, 40, 50, 60, 70 …160 bytes. That is the first TS packet adaption field is enabled and filled with N 0xff bytes. Validating output remains same as before test. These tests are necessary to check the ADTS frame header parsing logic inside ts2mfu code.

### Packing 2 frames in a TS packet

Make sure while creating set of audio TS packets, few TS packets has 2 frames (in whole or partial). This is to ensure that ADTS frame header parsing handles more than 1 frame in a TS packet.

### Packing 3 frames in a TS packet

Make sure while creating set of audio TS packets, few TS packets has 3 frames (in whole or partial). This is to ensure that ADTS frame header parsing handles more than 2 frames in a TS packet.