**CableLabs EBP Conformance Test Tool User Guide**

**Description**

The CableLabs EBP Conformance Test Tool is a command-line executable that ingests MPEG transport streams and tests their compliance with the ATS EBP specification. The Tool is written in C and runs on either Linux or Cygwin. Transport streams can be ingested either as files or as network multicast streams.

The main purpose of the Tool is to verify that the Encoder Boundary Points (EBP's) in separate transport streams are aligned in time. The tool does this by parsing the EBP descriptors and EBP structures in each transport stream, and then comparing the EBP PTS values across the different streams to validate that they are the same. Myriad other validations are performed, including:

* Validate the groups in the EBP structures.
* Check that audio EBP PTS lags video EBP PTS by no more than 3 seconds.
* Check that multiple EBP structures do not exist in a single PES packet.
* Check that EBP is not present on a TS packet that does not have PUSI bit set.
* Validate that EBP struct and EBP descriptors parse correctly.
* Verify that the acquisition time for a particular EBP is the same across transport streams.
* Verify that SCTE35 PTS values are consistent with EBP PTS values; the allowable jitter in this is configurable in the Tool.
* Verify that SAP Type in the stream is consistent with that in the EBP struct
* Verify that SAP Type in the EBP struct and sap\_type\_max in the relevant partition inthe EBP descriptor are consistent.
* If sap\_flag not present in the EBP struct, verify that the expected SAP Type is 1 or 2.
* Verify that the EBP PTS is consistent with the expected location as derived from the EBP descriptor's ebp\_distance ticks\_per\_second fields; the allowable jitter in this is configurable in the Tool.

The tool operates in the following way. First, an assessment is performed for each transport stream where the program streams and their accompanying EBP parameters are discovered. In the command line options to follow, this is called the “peek” phase. Among the EBP information discovered for each program stream are the EBP partitions present in each stream and whether those partitions are signaled explicitly or implicitly. The Tool handles all possible combinations of explicit/implicit EBP signaling discussed in the ATS specification (REF HERE).

After finishing the peek phase, the Tool begins the actual test on the transport streams. Multiple ingest threads, one per transport stream, are launched so that the test is performed in real-time. As the ingest threads find EBP’s, they hand off the EBP’s to analysis threads that do the comparison across transport streams. A summary result report is printed either at the end of the test (in the case of file-base transport streams), or at the request of the user (in the case of multicast transport streams).

**Operation**

The Tool ingests transport streams in two ways: as files, and as network multicast streams. For the file case, the command-line syntax is:

ATSTestApp –f [options] <input file 1> <input file 2> ... <input file N>

where the available options are:

-p: “peek” mode -- only perform initial diagnosis of stream (elementary streams, EBP descriptor info, etc); does not perform EBP validation.

The ATSTestApp then runs until the contents of all the files have been analyzed. A log (with a default name EBPTestLog.txt) is written, at the end of which is a pass/fail report on the findings. The log also contains detailed information on the EBP PTS’s found as well as on any errors encountered.

For the network multicast ingest case, the command-line syntax is:

ATSTestApp –m [options] [<source1>@]<ip1>:<port1>...[<sourceN>@]<ipN>:<portN>

where the available options are:

-p: “peek” mode -- only perform initial diagnosis of stream (elementary streams, EBP descriptor info, etc); does not perform EBP validation.

-d: save transport stream to file; file will be of the form EBPStreamDump\_IP.port.ts

The ATSTestApp opens sockets to receive the streams, and performs the “peek” analysis. After this is complete, the actual test begins. At this time the user is presented with a menu of options

x then return to exit

r then return to create report

c then return to clear report data

s then return to see a status of the incoming streams

The r option creates a report detailing the pass/fail status as well as details of any errors found. The c options deletes all test results and restarts the test. Finally, the s option shows the status of the various internal queues to check that the Tool is keeping up with the data being ingested. Note that if the peek phase is not completed, this menu will not appear.

**Configuration**

The Tool has various configurable properties contained in the file ATSTestApp.props. The default props file is:

// logLevel: 1 ERROR, 2 WARNING, 3 INFO, 4 DEBUG

logLevel = 3

// enter log path here then uncomment – default is EBPTestLog.txt

//logFilePath =

// amount of time spent searching for EBP structs at start of test in

// the case where EBP descriptor is not present

ebpPrereadSearchTimeMsecs = 10000

// allowed time difference between expected EBP location (from EBP

// descriptor, if present) actual EBP location

ebpAllowedPTSJitterSecs = 0.5

// allowed time difference between expected EBP location (from SCTE35,

// if present)and actual EBP location

ebpSCTE35PTSJitterSecs = 0.5

// for multicast case, size of UDP receive buffer

socketRcvBufferSz = 2000000

// size of buffer holding transport stream data waiting to be

// processed. This needs to be a bit larger than the

// ebpPrereadSearchTime above, since all of the data is cached here

// while it is analyzed.

ingestCircularBufferSz = 1880000

**Building**

The Tool can be built on either Linux or Windows (via Cygwin). To perform a build, go to the top-level directory and type “make”. The executable for the Tool is named ATSTestApp.exe, and resides in the atstest subfolder.

The subfolders contain source code for the various components:

atstest: contains top-level code and test threading and logic

tslib: contains transport packet parsing code

logging: contains logging implementation

libstructures: contains data structure implementation

h264bitstream: contains h264 parsing code

common: contains misc code

atsstreamapp: contains test utility to produce multicast streams from transport stream files.

**Debugging Issues**

The best approach for debugging issues is to open the log file (named by default EBPTestLog.txt, or alternatively specified in the props file), and search for ERROR entries.