UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Annex

Implementació d'un sistema de gestió remota de mescladors de vídeo ATEM

Autor: Lluc Costa Utges

Director: Juan Jose Alins DelgadoA

Annex 1

ATEMbase.cpp (Arduino)

```
/*
Copyright 2012-2014 Kasper Skårhøj, SKAARHOJ K/S, kasper@skaarhoj.com

This file is part of the Blackmagic Design ATEM Client library for Arduin
o

The ATEM library is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by the
Free Software Foundation, either version 3 of the License, or (at your
option) any later version.

The ATEM library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILIT
Y
or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with the ATEM library. If not, see http://www.gnu.org/licenses/.


IMPORTANT: If you want to use this library in your own projects and/or pr
oducts,
please play a fair game and heed the license rules! See our web page for
a Q&A so
you can keep a clear conscience: http://skaarhoj.com/about/licenses/


*/



#include "ATEMbase.h"



/**
 * Constructor
 */
ATEMbase::ATEMbase(){}
```

```
/**
 * Setting up IP address for the switcher (and local port to send packets
 from)
 * Using local port here is deprecated. Rather let the library pick a ran
dom one
 */
void ATEMbase::begin(const IPAddress ip){
    begin(ip, random(50100,65300));
}
void ATEMbase::begin(const IPAddress ip, const uint16_t localPort){

    neverConnected = true;
    waitingForIncoming = false;

        // Set up Udp communication object:
    #ifdef ESP8266
    WiFiUDP Udp;
    #else
    EthernetUDP Udp;
    #endif

    _Udp = Udp;

    _switcherIP = ip;           // Set switcher IP address
    _localPort = localPort;     // Set default local port

    _lastContact = 0;
    _serialOutput = 0;

    resetCommandBundle();
}

/**
 * Initiating connection handshake to the ATEM switcher
 */
void ATEMbase::connect() {
    connect(false);
}

/**
 * Initiating connection handshake to the ATEM switcher
 * If useFixedPortNumber is true, the same port number will be used on su
bsequent connects, otherwise - and recommended - a new, random port numbe
r is used.
 */
void ATEMbase::connect(const boolean useFixedPortNumber) {
    _localPacketIdCounter = 0;      // Init localPacketIDCounter to 0;
```

```
    _initPayloadSent = false;       // Will be true after initial payload
 of data is delivered (regular 12-byte ping packages are transmitted.)
    _hasInitialized = false;        // Will be true after initial payload
 of data is resent and received well
    _isConnected = false;           // Will be true after the initial hel
lo-package handshakes.
    _sessionID = 0x53AB;            // Temporary session ID - a new will
be given back from ATEM.
    _lastContact = millis();        // Setting this, because even though
we haven't had contact, it constitutes an attempt that should be responde
d to at least
    memset(_missedInitializationPackages, 0xFF, (ATEM_maxInitPackageCount
+7)/8);
    _initPayloadSentAtPacketId = ATEM_maxInitPackageCount;  // The max va
lue it can be
    uint16_t portNumber = useFixedPortNumber ? _localPort : random(50100,
65300);

    _Udp.begin(portNumber);


    // Send connectString to ATEM:
    if (_serialOutput)  {
        Serial.print(F("Sending connect packet to ATEM switcher on IP "))
;
        Serial.print(_switcherIP);
        Serial.print(F(" from port "));
        Serial.println(portNumber);
    }

    _wipeCleanPacketBuffer();
    _createCommandHeader(ATEM_headerCmd_HelloPacket, 12+8);
    _packetBuffer[12] = 0x01;   // This seems to be what the client shoul
d send upon first request.
    _packetBuffer[9] = 0x3a;    // This seems to be what the client shoul
d send upon first request.

    _sendPacketBuffer(20);
}

/**
 * Keeps connection to the switcher alive
 * Therefore: Call this in the Arduino loop() function and make sure it g
ets call at least 2 times a second
 * Other recommendations might come up in the future.
 */
void ATEMbase::runLoop() {
    runLoop(0);
}
```

```
void ATEMbase::runLoop(uint16_t delayTime) {
    if (neverConnected) {
        neverConnected = false;
        connect();
//      Serial.println("Connecting first time...");
    }

    unsigned long enterTime = millis();

    do {
        while(true) {    // Iterate until UDP buffer is empty
            uint16_t packetSize = _Udp.parsePacket();
            if (_Udp.available())   {
                _Udp.read(_packetBuffer,12);    // Read header
                 _sessionID = word(_packetBuffer[2], _packetBuffer[3]);
                 uint8_t headerBitmask = _packetBuffer[0]>>3;
                 _lastRemotePacketID = word(_packetBuffer[10],_packetBuff
er[11]);
                 if (_lastRemotePacketID < ATEM_maxInitPackageCount)    {
                    _missedInitializationPackages[_lastRemotePacketID>>3]
 &= ~(B1<<(_lastRemotePacketID&0x07));
                 }

                 uint16_t packetLength = word(_packetBuffer[0] & B0000011
1, _packetBuffer[1]);

                if (packetSize==packetLength) {  // Just to make sure the
se are equal, they should be!
                    _lastContact = millis();
                    waitingForIncoming = false;

                    if (headerBitmask & ATEM_headerCmd_HelloPacket) {   /
/ Respond to "Hello" packages:
                        _isConnected = true;

                        // _packetBuffer[12]    The ATEM will return a "2
" in this return package of same length. If the ATEM returns "3" it means
 "fully booked" (no more clients can connect) and a "4" seems to be a kin
d of reconnect (seen when you drop the connection and the ATEM desperatel
y tries to figure out what happened...)
                        // _packetBuffer[15]    This number seems to incr
ement with about 3 each time a new client tries to connect to ATEM. It ma
y be used to judge how many client connections has been made during the u
p-time of the switcher?

                        _wipeCleanPacketBuffer();
                        _createCommandHeader(ATEM_headerCmd_Ack, 12);
                        _packetBuffer[9] = 0x03;    // This seems to be w
hat the client should send upon first request.
```

```
                    _sendPacketBuffer(12);
                }

                // If a packet is 12 bytes long it indicates that all
 the initial information
                // has been delivered from the ATEM and we can begin
to answer back on every request
                // Currently we don't know any other way to decide if
 an answer should be sent back...
                // The QT lib uses the "InCm" command to indicate thi
s, but in the latest version of the firmware (2.14)
                // all the camera control information comes AFTER thi
s command, so it's not a clear ending token anymore.
                // However, I'm not sure if I checked the _lastRemote
PacketID of the packages with the additional camera control info - if it
was a resend,
                // "InCm" may still indicate the number of the last i
nit-package and that's all I need to request the missing ones....

                // BTW: It has been observed on an old 10Mbit hub tha
t packages could arrive in a different order than sent and this may
                // mess things up a bit on the initialization. So it'
s recommended to has as direct routes as possible.
                if(!_initPayloadSent && packetSize == 12 && _lastRemo
tePacketID>1) {
                    _initPayloadSent = true;
                    _initPayloadSentAtPacketId = _lastRemotePacketID;
                    #if ATEM_debug
                    if (_serialOutput & 0x80) {
                        Serial.print(F("_initPayloadSent=TRUE @rpID "
));
                        Serial.println(_initPayloadSentAtPacketId);
                        Serial.print(F("Session ID: "));
                        Serial.println(_sessionID, DEC);
                    }
                    #endif
                }

                if (_initPayloadSent && (headerBitmask & ATEM_headerC
md_AckRequest) && (_hasInitialized || !(headerBitmask & ATEM_headerCmd_Re
send))) {      // Respond to request for acknowledge   (and to resends als
o, whatever...
                    _wipeCleanPacketBuffer();
                    _createCommandHeader(ATEM_headerCmd_Ack, 12, _las
tRemotePacketID);
                    _sendPacketBuffer(12);

                    #if ATEM_debug
                    if (_serialOutput & 0x80) {
```

```
                                        Serial.print(F("rpID: "));
                                        Serial.print(_lastRemotePacketID, DEC);
                                        Serial.print(F(", Head: 0x"));
                                        Serial.print(headerBitmask, HEX);
                                        Serial.print(F(", Len: "));
                                        Serial.print(packetLength, DEC);
                                        Serial.print(F(" bytes"));

                                        Serial.println(F(" - ACK!"));
                                    } else
                                    #endif
                                    if (_serialOutput>1)    {
                                        Serial.print(F("rpID: "));
                                        Serial.print(_lastRemotePacketID, DEC);
                                        Serial.println(F(" - ACK!"));
                                    }
                                } else if(_initPayloadSent && (headerBitmask & ATEM_h
eaderCmd_RequestNextAfter) && _hasInitialized) {   // ATEM is requesting
a previously sent package which must have dropped out of the order. We re
turn an empty one so the ATEM doesnt' crash (which some models will, if i
t doesn't get an answer before another 63 commands gets sent from the con
troller.)
                                    uint8_t b1 = _packetBuffer[6];
                                    uint8_t b2 = _packetBuffer[7];
                                    _wipeCleanPacketBuffer();
                                    _createCommandHeader(ATEM_headerCmd_Ack, 12, 0);
                                    _packetBuffer[0] = ATEM_headerCmd_AckRequest << 3
;  // Overruling this. A small trick because createCommandHeader shouldn'
t increment local package ID counter
                                    _packetBuffer[10] = b1;
                                    _packetBuffer[11] = b2;
                                    _sendPacketBuffer(12);

                                    if (_serialOutput>1)    {
                                        Serial.print(F("ATEM asking to resend "));
                                        Serial.println((b1<<8)|b2, DEC);
                                    }
                                } else {
                                    #if ATEM_debug
                                    if (_serialOutput & 0x80) {
                                        Serial.print(F("rpID: "));
                                        Serial.print(_lastRemotePacketID, DEC);
                                        Serial.print(F(", Head: 0x"));
                                        Serial.print(headerBitmask, HEX);
                                        Serial.print(F(", Len: "));
                                        Serial.print(packetLength, DEC);
                                        Serial.println(F(" bytes"));
                                    } else
                                    #endif
```

```
                        if (_serialOutput>1)    {
                            Serial.print(F("rpID: "));
                            Serial.println(_lastRemotePacketID, DEC);
                        }
                    }

                    if (!(headerBitmask & ATEM_headerCmd_HelloPacket) &&
packetLength>12)    {
                        _parsePacket(packetLength);
                    }
                } else {
                    #if ATEM_debug
                    if (_serialOutput & 0x80)   {
                        Serial.print(F("ERROR: Packet size mismatch: "));
                        Serial.print(packetSize, DEC);
                        Serial.print(F(" != "));
                        Serial.println(packetLength, DEC);
                    }
                    #endif
                    // Flushing:
                    while(_Udp.available()) {
                        _Udp.read(_packetBuffer, ATEM_packetBufferLength)
;
                    }
                }
            } else break;
        }

        // After initialization, we check which packages were missed and
ask for them:
        if (!_hasInitialized && _initPayloadSent && !waitingForIncoming)
   {
            for(uint8_t i=1; i<_initPayloadSentAtPacketId; i++) {
                if (_missedInitializationPackages[i>>3] & (B1<<(i & 0x7))
)  {

                    #if ATEM_debug
                    if (_serialOutput & 0x80)   {
                        Serial.print(F("Asking for package "));
                        Serial.println(i, DEC);
                    }
                    #endif
                    _wipeCleanPacketBuffer();
                    _createCommandHeader(ATEM_headerCmd_RequestNextAfter,
 12);
                    _packetBuffer[6] = highByte(i-
1);  // Resend Packet ID, MSB
                    _packetBuffer[7] = lowByte(i-
1);  // Resend Packet ID, LSB
```

```
                    _packetBuffer[8] = 0x01;

                    _sendPacketBuffer(12);
                    waitingForIncoming = true;
                    break;
                }
            }
            if (!waitingForIncoming)    {
                _hasInitialized = true;
                if (_serialOutput) {
                    Serial.println(F("ATEM _hasInitialized = TRUE"));
                }
            }
        }
    } while (delayTime>0 && !hasTimedOut(enterTime,delayTime));


    // If connection is gone anyway, try to reconnect:
    if (hasTimedOut(_lastContact, 5000))    {
      if (_serialOutput) Serial.println(F("Connection to ATEM Switcher ha
s timed out - reconnecting!"));
      connect();
    }
}

/**
 * Returns last Remote Packet ID
 */
uint16_t ATEMbase::getATEM_lastRemotePacketId() {
    return _lastRemotePacketID;
}

/**
 * Get ATEM session ID
 */
uint16_t ATEMbase::getSessionID() {
    return _sessionID;
}

/**
 * If true, we had a response from the switcher when trying to send a hel
lo packet.
 */
bool ATEMbase::isConnected()    {
    return _isConnected;
}

/**
```

```
 * If true, the initial handshake and "stressful" information exchange ha
s occured and now the switcher connection should be ready for operation.
 */
bool ATEMbase::hasInitialized() {
    return _hasInitialized;
}




/**************
 *
 * Buffer work
 *
 **************/

void ATEMbase::_createCommandHeader(const uint8_t headerCmd, const uint16
_t lengthOfData)   {
    _createCommandHeader(headerCmd, lengthOfData, 0);
}
void ATEMbase::_createCommandHeader(const uint8_t headerCmd, const uint16
_t lengthOfData, const uint16_t remotePacketID)    {

    _packetBuffer[0] = (headerCmd << 3) | (highByte(lengthOfData) & 0x07)
;  // Command bits + length MSB
    _packetBuffer[1] = lowByte(lengthOfData);  // length LSB

    _packetBuffer[2] = highByte(_sessionID);  // Session ID
    _packetBuffer[3] = lowByte(_sessionID);  // Session ID

    _packetBuffer[4] = highByte(remotePacketID);  // Remote Packet ID, MS
B
    _packetBuffer[5] = lowByte(remotePacketID);  // Remote Packet ID, LSB

    if(!(headerCmd & (ATEM_headerCmd_HelloPacket | ATEM_headerCmd_Ack | A
TEM_headerCmd_RequestNextAfter))) {
        _localPacketIdCounter++;

//      if ((_localPacketIdCounter & 0xF) == 0xF) _localPacketIdCounter++
;  // Uncommenting this line will jump the local package ID counter every
 15 command - thereby introducing a stress test of the robustness of the
"resent package" function from the ATEM switcher.

        _packetBuffer[10] = highByte(_localPacketIdCounter);  // Local Pa
cket ID, MSB
        _packetBuffer[11] = lowByte(_localPacketIdCounter);  // Local Pac
ket ID, LSB
```

```
    }
}
void ATEMbase::_sendPacketBuffer(uint8_t length)    {
    _Udp.beginPacket(_switcherIP,  9910);
    _Udp.write(_packetBuffer,length);
    _Udp.endPacket();   // TODO: Figure out why this may hang!!
}

/**
 * Sets all zeros in packet buffer:
 */
void ATEMbase::_wipeCleanPacketBuffer() {
    memset(_packetBuffer, 0, ATEM_packetBufferLength);
}

/**
 * Reads from UDP channel to buffer. Will fill the buffer to the max or t
o the size of the current segment being parsed
 * Returns false if there are no more bytes, otherwise true
 */
bool ATEMbase::_readToPacketBuffer() {
    return _readToPacketBuffer(ATEM_packetBufferLength);
}
bool ATEMbase::_readToPacketBuffer(uint8_t maxBytes) {
    maxBytes = maxBytes<=ATEM_packetBufferLength ? maxBytes : ATEM_packet
BufferLength;
    int remainingBytes = _cmdLength-8-_cmdPointer;

    if (remainingBytes>0)   {
        if (remainingBytes <= maxBytes) {
            _Udp.read(_packetBuffer, remainingBytes);
            _cmdPointer+= remainingBytes;
            return false;   // Returns false if finished.
        } else {
            _Udp.read(_packetBuffer, maxBytes);
            _cmdPointer+= maxBytes;
            return true;    // Returns true if there are still bytes to b
e read.
        }
    } else {
        return false;
    }
}

/**
 * If a package longer than a normal acknowledgement is received from the
 ATEM Switcher we must read through the contents.
 * Usually such a package contains updated state information about the mi
xer
```

```
 * Selected information is extracted in this function and transferred to
internal variables in this library.
 */
void ATEMbase::_parsePacket(uint16_t packetLength)  {

        // If packet is more than an ACK packet (= if its longer than 12
bytes header), lets parse it:
      uint16_t indexPointer = 12;   // 12 bytes has already been read fro
m the packet...
      while (indexPointer < packetLength)  {

        // Read the length of segment (first word):
        _Udp.read(_packetBuffer, 8);
        _cmdLength = word(_packetBuffer[0], _packetBuffer[1]);
        _cmdPointer = 0;

            // Get the "command string", basically this is the 4 char var
iable name in the ATEM memory holding the various state values of the sys
tem:
        char cmdStr[] = {
          _packetBuffer[4], _packetBuffer[5], _packetBuffer[6], _packetBu
ffer[7], '\0'};

            // If length of segment larger than 8 (should always be...!)
        if (_cmdLength>8)  {
            _parseGetCommands(cmdStr);

            while (_readToPacketBuffer())   {}  // Empty, if not done yet
.

            indexPointer+=_cmdLength;
        } else {
            indexPointer = 2000;
            #if ATEM_debug
                if (_serialOutput & 0x80) Serial.println(F("Bad CMD lengt
h, flushing..."));
            #endif

            // Flushing the buffer:
              while(_Udp.available()) {
                  _Udp.read(_packetBuffer, ATEM_packetBufferLength);
              }
        }
      }
}

/**
 * This method should be overloaded in subclasses in order to handle spec
ific get-commands
 */
```

```
void ATEMbase::_parseGetCommands(const char *cmdString) {
//  uint8_t mE, keyer, mediaPlayer, aUXChannel, windowIndex, multiViewer,
 memory, colorGenerator, box;
//  uint16_t audioSource, videoSource;
//  long temp;

    uint8_t numberOfReads=1;
    while(_readToPacketBuffer())    {
        numberOfReads++;
    }

    #if ATEM_debug
    if (_serialOutput & 0x80) {
        Serial.print(cmdString);
        Serial.print(", len: ");
        Serial.print(_cmdLength);
        Serial.print(", rds: ");
        Serial.println(numberOfReads);
    }
    #endif
}

void ATEMbase::_prepareCommandPacket(const char *cmdString, uint8_t cmdBy
tes, bool indexMatch)  {

        // First, in case of a command bundle, check if indexes are diffe
rent OR if it's an entirely different command, then increase offset to ac
commodate new command:
        if (_cBundle)   {
            if (_returnPacketLength>0 && (!indexMatch || strncmp_P((char
*)(_packetBuffer+12+_cBBO+4), cmdString, 4)))  {
                _cBBO = _returnPacketLength-12;
            }
        } else {
            _wipeCleanPacketBuffer();   // For command bundles, this is a
lready done...
        }

      _returnPacketLength = 12+_cBBO+(4+4+cmdBytes);

      // Because we increased length of command, we need to check for buf
fer overflow:
      if (_returnPacketLength > ATEM_packetBufferLength)    {
          Serial.println(F("FATAL ERROR: Packet Buffer Overflow in the AT
EM Library! Too long or too many commands bundled!\n HALT"));
          while(true){} // STOP!
      }

        // Copy Command String:
```

```
        if (strlen_P(cmdString)==4) {
            strncpy_P((char *)(_packetBuffer+12+_cBBO+4), cmdString, 4);
        }
        #if ATEM_debug
        else Serial.println(F("Command Length > 4 ERROR"));
        #endif

      // Command length:
      _packetBuffer[12+_cBBO] = 0;  // MSB - but it's always under 256, s
o....
      _packetBuffer[12+1+_cBBO] = 4+4+cmdBytes; // LSB
}

void ATEMbase::_finishCommandPacket()   {
    if (!_cBundle)  {

      _createCommandHeader(ATEM_headerCmd_AckRequest, _returnPacketLength
);
      _sendPacketBuffer(_returnPacketLength);
      _returnPacketLength = 0;

    } else {
          // Debugging info:
    /*    for(uint8_t a=0; a<_returnPacketLength; a++)  {
          if (_packetBuffer[a]<16)  Serial.print("0");
              Serial.print(_packetBuffer[a], HEX);
              Serial.print(F("-"));
          }
          Serial.println();
        */
    }
}



/*************
 *
 * Utilities from SkaarhojTools class:
 *
 *************/

/**
 * Setter method: If _serialOutput is set, the library may use Serial.pri
nt() to give away information about its operation - mostly for debugging.
 * 0= no output
 * 1= normal output (info)
 * 2= verbose
 * &0x80 (bit 7 set): verbose initial connection information
 */
```

```
void ATEMbase::serialOutput(uint8_t level) {
    _serialOutput = level;
}

/**
 * Timeout check
 */
bool ATEMbase::hasTimedOut(unsigned long time, unsigned long timeout)  {
  if ((unsigned long)(time + timeout) <= (unsigned long)millis())  {  //
This should "wrap around" if time+timout is larger than the size of unsig
ned-longs, right?
    return true;
  }
  else {
    return false;
  }
}




uint8_t ATEMbase::getATEMmodel()    {
    return _ATEMmodel;
}




float ATEMbase::audioWord2Db(uint16_t input)  {  // -48 to +6 output
  // Formular: log10(input/128)*20-48;
  if (input<=32)  return -60;
  //return (log10(input)-2.1072099696)*20-48;
  // Better way?
  //return log10(input >> 5) * 20.0 - 60.0;

  return log10((float)input/(1<<11) / 16.0) * 20.0;
}
uint16_t ATEMbase::audioDb2Word(float input)  {  // -48 to +6 input
  //return (float)pow(10,(input+48)/20)*128;
    //return (uint16_t)pow(10, (input + 60.0) / 20.0) << 5;

    return pow(10, input/20.0) * 16.0 * (1<<11);
}
```

```
uint8_t ATEMbase::getVideoSrcIndex(uint16_t videoSrc)   {
    switch(videoSrc){
        case 0:  // Black
            return 0;
        case 1:  // Input 1
            return 1;
        case 2:  // Input 2
            return 2;
        case 3:  // Input 3
            return 3;
        case 4:  // Input 4
            return 4;
        case 5:  // Input 5
            return 5;
        case 6:  // Input 6
            return 6;
        case 7:  // Input 7
            return 7;
        case 8:  // Input 8
            return 8;
        case 9:  // Input 9
            return 9;
        case 10:  // Input 10
            return 10;
        case 11:  // Input 11
            return 11;
        case 12:  // Input 12
            return 12;
        case 13:  // Input 13
            return 13;
        case 14:  // Input 14
            return 14;
        case 15:  // Input 15
            return 15;
        case 16:  // Input 16
            return 16;
        case 17:  // Input 17
            return 17;
        case 18:  // Input 18
            return 18;
        case 19:  // Input 19
            return 19;
```

```
case 20:  // Input 20
    return 20;
case 1000:  // Color Bars
    return 21;
case 2001:  // Color 1
    return 22;
case 2002:  // Color 2
    return 23;
case 3010:  // Media Player 1
    return 24;
case 3011:  // Media Player 1 Key
    return 25;
case 3020:  // Media Player 2
    return 26;
case 3021:  // Media Player 2 Key
    return 27;
case 4010:  // Key 1 Mask
    return 28;
case 4020:  // Key 2 Mask
    return 29;
case 4030:  // Key 3 Mask
    return 30;
case 4040:  // Key 4 Mask
    return 31;
case 5010:  // DSK 1 Mask
    return 32;
case 5020:  // DSK 2 Mask
    return 33;
case 6000:  // Super Source
    return 34;
case 7001:  // Clean Feed 1
    return 35;
case 7002:  // Clean Feed 2
    return 36;
case 8001:  // Auxilary 1
    return 37;
case 8002:  // Auxilary 2
    return 38;
case 8003:  // Auxilary 3
    return 39;
case 8004:  // Auxilary 4
    return 40;
case 8005:  // Auxilary 5
    return 41;
case 8006:  // Auxilary 6
    return 42;
case 10010:  // ME 1 Prog
    return 43;
case 10011:  // ME 1 Prev
```

```
            return 44;
        case 10020:  // ME 2 Prog
            return 45;
        case 10021:  // ME 2 Prev
            return 46;
        default:
            return 0;
    }
}

uint8_t ATEMbase::getAudioSrcIndex(uint16_t audioSrc)   {
    switch(audioSrc){
        case 1:  // Input 1
            return 0;
        case 2:  // Input 2
            return 1;
        case 3:  // Input 3
            return 2;
        case 4:  // Input 4
            return 3;
        case 5:  // Input 5
            return 4;
        case 6:  // Input 6
            return 5;
        case 7:  // Input 7
            return 6;
        case 8:  // Input 8
            return 7;
        case 9:  // Input 9
            return 8;
        case 10:  // Input 10
            return 9;
        case 11:  // Input 11
            return 10;
        case 12:  // Input 12
            return 11;
        case 13:  // Input 13
            return 12;
        case 14:  // Input 14
            return 13;
        case 15:  // Input 15
            return 14;
        case 16:  // Input 16
            return 15;
        case 17:  // Input 17
            return 16;
        case 18:  // Input 18
            return 17;
        case 19:  // Input 19
```

```
            return 18;
        case 20:  // Input 20
            return 19;
        case 1001:  // XLR
            return 20;
        case 1101:  // AES/EBU
            return 21;
        case 1201:  // RCA
            return 22;
        case 2001:  // MP1
            return 23;
        case 2002:  // MP2
            return 24;
        default:
            return 0;
    }
}

/*
 * Translating a index to a video source
 */
uint16_t ATEMbase::getVideoIndexSrc(uint8_t index)  {
  switch (index) {
    case 0:  // Black
      return 0;
    case 1:  // Input 1
      return 1;
    case 2:  // Input 2
      return 2;
    case 3:  // Input 3
      return 3;
    case 4:  // Input 4
      return 4;
    case 5:  // Input 5
      return 5;
    case 6:  // Input 6
      return 6;
    case 7:  // Input 7
      return 7;
    case 8:  // Input 8
      return 8;
    case 9:  // Input 9
      return 9;
    case 10:  // Input 10
      return 10;
    case 11:  // Input 11
      return 11;
    case 12:  // Input 12
      return 12;
```

```
case 13:  // Input 13
  return 13;
case 14:  // Input 14
  return 14;
case 15:  // Input 15
  return 15;
case 16:  // Input 16
  return 16;
case 17:  // Input 17
  return 17;
case 18:  // Input 18
  return 18;
case 19:  // Input 19
  return 19;
case 20:  // Input 20
  return 20;
case 21:  // Color Bars
  return 1000;
case 22:  // Color 1
  return 2001;
case 23:  // Color 2
  return 2002;
case 24:  // Media Player 1
  return 3010;
case 25:  // Media Player 1 Key
  return 3011;
case 26:  // Media Player 2
  return 3020;
case 27:  // Media Player 2 Key
  return 3021;
case 28:  // Key 1 Mask
  return 4010;
case 29:  // Key 2 Mask
  return 4020;
case 30:  // Key 3 Mask
  return 4030;
case 31:  // Key 4 Mask
  return 4040;
case 32:  // DSK 1 Mask
  return 5010;
case 33:  // DSK 2 Mask
  return 5020;
case 34:  // Super Source
  return 6000;
case 35:  // Clean Feed 1
  return 7001;
case 36:  // Clean Feed 2
  return 7002;
case 37:  // Auxilary 1
```

```
      return 8001;
    case 38:  // Auxilary 2
      return 8002;
    case 39:  // Auxilary 3
      return 8003;
    case 40:  // Auxilary 4
      return 8004;
    case 41:  // Auxilary 5
      return 8005;
    case 42:  // Auxilary 6
      return 8006;
    case 43:  // ME 1 Prog
      return 10010;
    case 44:  // ME 1 Prev
      return 10011;
    case 45:  // ME 2 Prog
      return 10020;
    case 46:  // ME 2 Prev
      return 10021;
    default:
      return 0;
  }
}

/*
 * Translating a index to a audio source
 */
uint16_t ATEMbase::getAudioIndexSrc(uint8_t index)  {
  switch (index) {
    case 0:  // Input 1
      return 1;
    case 1:  // Input 2
      return 2;
    case 2:  // Input 3
      return 3;
    case 3:  // Input 4
      return 4;
    case 4:  // Input 5
      return 5;
    case 5:  // Input 6
      return 6;
    case 6:  // Input 7
      return 7;
    case 7:  // Input 8
      return 8;
    case 8:  // Input 9
      return 9;
    case 9:  // Input 10
      return 10;
```

```
    case 10:  // Input 11
      return 11;
    case 11:  // Input 12
      return 12;
    case 12:  // Input 13
      return 13;
    case 13:  // Input 14
      return 14;
    case 14:  // Input 15
      return 15;
    case 15:  // Input 16
      return 16;
    case 16:  // Input 17
      return 17;
    case 17:  // Input 18
      return 18;
    case 18:  // Input 19
      return 19;
    case 19:  // Input 20
      return 20;
    case 20:  // XLR
      return 1001;
    case 21:  // AES/EBU
      return 1101;
    case 22:  // RCA
      return 1201;
    case 23:  // MP1
      return 2001;
    case 24:  // MP2
      return 2002;
    default:
      return 0;
  }
}

uint8_t ATEMbase::maxAtemSeriesVideoInputs()    {
    return 47;  // For the largest ATEM switcher, this is the number of v
ideo inputs. The max "index" number from the list above
}


void ATEMbase::commandBundleStart() {
    resetCommandBundle();
    _wipeCleanPacketBuffer();
    _cBundle = true;
}
void ATEMbase::commandBundleEnd()   {
    if (_cBundle && _returnPacketLength > 0)    {
```

```
        _createCommandHeader(ATEM_headerCmd_AckRequest, _returnPacketLength
);
        _sendPacketBuffer(_returnPacketLength);
        _returnPacketLength = 0;
    }
    resetCommandBundle();
}
void ATEMbase::resetCommandBundle() {
    _cBundle = false;
    _cBBO = 0;
}
```

Annex 2

ATEMstd.cpp (Arduino)

```
*
Copyright 2012-2014 Kasper Skårhøj, SKAARHOJ K/S, kasper@skaarhoj.com

This file is part of the Blackmagic Design ATEM Client library for Arduin
o

The ATEM library is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by the
Free Software Foundation, either version 3 of the License, or (at your
option) any later version.

The ATEM library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILIT
Y
or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with the ATEM library. If not, see http://www.gnu.org/licenses/.


IMPORTANT: If you want to use this library in your own projects and/or pr
oducts,
please play a fair game and heed the license rules! See our web page for
a Q&A so
you can keep a clear conscience: http://skaarhoj.com/about/licenses/


*/



#include "ATEMstd.h"


/**
 * Constructor (using arguments is deprecated! Use begin() instead)
 */
```

```
ATEMstd::ATEMstd(){}


void ATEMstd::delay(const unsigned int delayTimeMillis) {   // Responsibl
e delay function which keeps the ATEM run loop up! DO NOT USE INSIDE THIS
 CLASS! Recursion could happen...
    runLoop(delayTimeMillis);
}




/********************************
 *
 * ATEM Switcher state methods
 * Returns the most recent information we've
 * got about the switchers state
 *
 ********************************/

uint16_t ATEMstd::getProgramInput() {
    return getProgramInputVideoSource(0);
}
uint16_t ATEMstd::getPreviewInput() {
    return getPreviewInputVideoSource(0);
}
boolean ATEMstd::getProgramTally(uint8_t inputNumber) {
    return (getTallyByIndexTallyFlags(inputNumber-
1) & 1) >0 ? true : false;
}
boolean ATEMstd::getPreviewTally(uint8_t inputNumber) {
    return (getTallyByIndexTallyFlags(inputNumber-
1) & 2) >0 ? true : false;
}
boolean ATEMstd::getUpstreamKeyerStatus(uint8_t inputNumber) {
    return getKeyerOnAirEnabled(0,inputNumber-1);
}
boolean ATEMstd::getUpstreamKeyerOnNextTransitionStatus(uint8_t inputNumb
er) {  // input 0 = background
    return (getTransitionNextTransition(0) & (0x01 << inputNumber)) ? tru
e : false;
}
boolean ATEMstd::getDownstreamKeyerStatus(uint8_t inputNumber) {
    return getDownstreamKeyerOnAir(inputNumber-1);
}
uint16_t ATEMstd::getTransitionPosition() {
    return getTransitionPosition(0);
}
```

```
bool ATEMstd::getTransitionPreview()    {
    return getTransitionPreviewEnabled(0);
}
uint8_t ATEMstd::getTransitionType()    {
    return getTransitionStyle(0);
}
uint8_t ATEMstd::getTransitionMixTime() {
    return getTransitionMixRate(0);     // Transition time for Mix Transi
tions
}
boolean ATEMstd::getFadeToBlackState() {
    return getFadeToBlackStateFullyBlack(0);    // Active state of Fade-
to-black
}
uint8_t ATEMstd::getFadeToBlackFrameCount() {
    return getFadeToBlackStateFramesRemaining(0);    // Returns current f
rame in the FTB
}
uint8_t ATEMstd::getFadeToBlackTime() {
    return getFadeToBlackRate(0);       // Transition time for Fade-to-
black
}
bool ATEMstd::getDownstreamKeyTie(uint8_t keyer)    {
    return getDownstreamKeyerTie(keyer-1);
}
uint16_t ATEMstd::getAuxState(uint8_t auxOutput)  {
    return getAuxSourceInput(auxOutput-1);
}
uint8_t ATEMstd::getMediaPlayerType(uint8_t mediaPlayer)  {
    return getMediaPlayerSourceType(mediaPlayer-1);
}
uint8_t ATEMstd::getMediaPlayerStill(uint8_t mediaPlayer)  {
    return getMediaPlayerSourceStillIndex(mediaPlayer-1)+1;
}
uint8_t ATEMstd::getMediaPlayerClip(uint8_t mediaPlayer)  {
    return getMediaPlayerSourceClipIndex(mediaPlayer-1)+1;
}
uint16_t ATEMstd::getAudioLevels(uint8_t channel)   {
    if (channel>0)  {
        return atemAudioMixerLevelsSourceRight;
    } else {
        return atemAudioMixerLevelsSourceLeft;
    }
}
uint8_t ATEMstd::getAudioChannelMode(uint16_t channelNumber)    {
    return getAudioMixerInputMixOption(channelNumber);
}
```

```
/******************************
 *
 * ATEM Switcher Change methods
 * Asks the switcher to changes something
 *
 ******************************/



void ATEMstd::changeProgramInput(uint16_t inputNumber)  {
    setProgramInputVideoSource(0, inputNumber);
}
void ATEMstd::changePreviewInput(uint16_t inputNumber)  {
    setPreviewInputVideoSource(0, inputNumber);
}
void ATEMstd::doCut()   {
    performCutME(0);
}
void ATEMstd::doAuto()  {
    performAutoME(0);
}
void ATEMstd::doAuto(uint8_t me)    {
    performAutoME(me);
}
void ATEMstd::fadeToBlackActivate() {
    performFadeToBlackME(0);
}
void ATEMstd::changeTransitionPosition(word value)  {
    setTransitionPosition(0,value);
}
void ATEMstd::changeTransitionPositionDone()    {   // When the last valu
e of the transition is sent (1000), send this one too (we are done, chang
e tally lights and preview bus!)
    setTransitionPosition(0,0);
}
void ATEMstd::changeTransitionPreview(bool state)   {
    setTransitionPreviewEnabled(0, state);
}
void ATEMstd::changeTransitionType(uint8_t type)    {
    setTransitionStyle(0, type);
}
void ATEMstd::changeTransitionMixTime(uint8_t frames)   {
    setTransitionMixRate(0, frames);
}
void ATEMstd::changeFadeToBlackTime(uint8_t frames) {
    setFadeToBlackRate(0, frames);
}
void ATEMstd::changeUpstreamKeyOn(uint8_t keyer, bool state)    {
    setKeyerOnAirEnabled(0, keyer-1, state);
```

```
}
void ATEMstd::changeUpstreamKeyNextTransition(uint8_t keyer, bool state)
   {   // Supporting "Background" by "0"
    if (keyer>=0 && keyer<=4)   {   // Todo: Should match available keyer
s depending on model?
        uint8_t stateValue = getTransitionNextTransition(0);
        if (state)  {
            stateValue = stateValue | (B1 << keyer);
        } else {
            stateValue = stateValue & (~(B1 << keyer));
        }
        setTransitionNextTransition(0, stateValue);
    }
}
void ATEMstd::changeDownstreamKeyOn(uint8_t keyer, bool state)  {
    setDownstreamKeyerOnAir(keyer-1, state);
}
void ATEMstd::changeDownstreamKeyTie(uint8_t keyer, bool state) {
    setDownstreamKeyerTie(keyer-1, state);
}
void ATEMstd::doAutoDownstreamKeyer(uint8_t keyer)  {
    performDownstreamKeyerAutoKeyer(keyer-1);
}
void ATEMstd::changeAuxState(uint8_t auxOutput, uint16_t inputNumber)  {
    setAuxSourceInput(auxOutput-1, inputNumber);
}
void ATEMstd::settingsMemorySave()  {
    _prepareCommandPacket(PSTR("SRsv"),4);
    _finishCommandPacket();
}
void ATEMstd::settingsMemoryClear() {
    _prepareCommandPacket(PSTR("SRcl"),4);
    _finishCommandPacket();
}
void ATEMstd::changeColorValue(uint8_t colorGenerator, uint16_t hue, uint
16_t saturation, uint16_t lightness)  {
    commandBundleStart();
    setColorGeneratorHue(colorGenerator-1, hue);
    setColorGeneratorSaturation(colorGenerator-1, saturation);
    setColorGeneratorLuma(colorGenerator-1, lightness);
    commandBundleEnd();
}
void ATEMstd::mediaPlayerSelectSource(uint8_t mediaPlayer, boolean moviec
lip, uint8_t sourceIndex)  {
    if (movieclip)  {
        commandBundleStart();
        setMediaPlayerSourceType(mediaPlayer-1, 2);
        setMediaPlayerSourceClipIndex(mediaPlayer-1,sourceIndex-1);
        commandBundleEnd();
```

```
    } else {
        commandBundleStart();
        setMediaPlayerSourceType(mediaPlayer-1, 1);
        setMediaPlayerSourceStillIndex(mediaPlayer-1,sourceIndex-1);
        commandBundleEnd();
    }
}

void ATEMstd::mediaPlayerClipStart(uint8_t mediaPlayer)  {
    setClipPlayerPlaying(mediaPlayer-1,true);
}

void ATEMstd::changeSwitcherVideoFormat(uint8_t format) {
    setVideoModeFormat(format);
}



void ATEMstd::changeDVESettingsTemp(unsigned long Xpos,unsigned long Ypos
,unsigned long Xsize,unsigned long Ysize)  {   // TEMP
    commandBundleStart();
    setKeyDVEPositionX(0, 0, Xpos);
    setKeyDVEPositionY(0, 0, Ypos);
    setKeyDVESizeX(0, 0, Xsize);
    setKeyDVESizeY(0, 0, Ysize);
    commandBundleEnd();
}
void ATEMstd::changeDVEMaskTemp(unsigned long top,unsigned long bottom,un
signed long left,unsigned long right)  {   // TEMP
    // N/A
}
void ATEMstd::changeDVEBorder(bool enableBorder)    {   // TEMP
    setKeyDVEBorderEnabled(0, 0, enableBorder);
}

void ATEMstd::changeDVESettingsTemp_Rate(uint8_t rateFrames)    {   // TE
MP
    setKeyDVERate(0,0,rateFrames);
}
void ATEMstd::changeDVESettingsTemp_RunKeyFrame(uint8_t runType)    {   /
/ runType: 1=A, 2=B, 3=Full, 4=on of the others (with an extra paramter:)
    setRunFlyingKeyRuntoInfiniteindex(0,0, runType);
}
void ATEMstd::changeKeyerMask(uint16_t topMask, uint16_t bottomMask, uint
16_t leftMask, uint16_t rightMask) {
    changeKeyerMask(0, topMask, bottomMask, leftMask, rightMask);
}
void ATEMstd::changeKeyerMask(uint8_t keyer, uint16_t topMask, uint16_t b
ottomMask, uint16_t leftMask, uint16_t rightMask)  {
```

```
    commandBundleStart();
    setKeyerTop(0, keyer, topMask);
    setKeyerBottom(0, keyer, bottomMask);
    setKeyerLeft(0, keyer, leftMask);
    setKeyerRight(0, keyer, rightMask);
    commandBundleEnd();
}

void ATEMstd::changeDownstreamKeyMask(uint8_t keyer, uint16_t topMask, ui
nt16_t bottomMask, uint16_t leftMask, uint16_t rightMask)  {
        if (keyer>=1 && keyer<=2)   {
            commandBundleStart();
            setDownstreamKeyerTop(keyer, topMask);
            setDownstreamKeyerBottom(keyer, bottomMask);
            setDownstreamKeyerLeft(keyer, leftMask);
            setDownstreamKeyerRight(keyer, rightMask);
            commandBundleEnd();
        }
}



void ATEMstd::changeUpstreamKeyFillSource(uint8_t keyer, uint16_t inputNu
mber)  {
    if (keyer>=1 && keyer<=4)   {   // Todo: Should match available keyer
s depending on model?
        setKeyerFillSource(0, keyer, inputNumber);
    }
}

// TODO: ONLY clip works right now! there is a bug...
void ATEMstd::changeUpstreamKeyBlending(uint8_t keyer, bool preMultiplied
Alpha, uint16_t clip, uint16_t gain, bool invKey)  {
    if (keyer>=1 && keyer<=4)   {   // Todo: Should match available keyer
s depending on model?
        commandBundleStart();
        setKeyLumaPreMultiplied(0, keyer, preMultipliedAlpha);
        setKeyLumaClip(0, keyer, clip);
        setKeyLumaGain(0, keyer, gain);
        setKeyLumaInvertKey(0, keyer, invKey);
        commandBundleEnd();
    }
}

// TODO: ONLY clip works right now! there is a bug...
void ATEMstd::changeDownstreamKeyBlending(uint8_t keyer, bool preMultipli
edAlpha, uint16_t clip, uint16_t gain, bool invKey)    {
    if (keyer>=1 && keyer<=2)   {   // Todo: Should match available keyer
s depending on model?
```

```
        commandBundleStart();
        setDownstreamKeyerPreMultiplied(keyer, preMultipliedAlpha);
        setDownstreamKeyerClip(keyer, clip);
        setDownstreamKeyerGain(keyer, gain);
        setDownstreamKeyerInvertKey(keyer, invKey);
        commandBundleEnd();
    }
}

// Statuskode retur: DskB, data byte 2 derefter er fill source, data byte
 3 er key source, data byte 1 er keyer 1-2 (0-1)
// Key source command er : CDsC - og ellers ens med...
void ATEMstd::changeDownstreamKeyFillSource(uint8_t keyer, uint16_t input
Number)    {
    if (keyer>=1 && keyer<=2)   {   // Todo: Should match available keyer
s depending on model?
        setDownstreamKeyerFillSource(keyer, inputNumber);
    }
}

void ATEMstd::changeDownstreamKeyKeySource(uint8_t keyer, uint16_t inputN
umber) {
    if (keyer>=1 && keyer<=2)   {   // Todo: Should match available keyer
s depending on model?
        setDownstreamKeyerKeySource(keyer, inputNumber);
    }
}



void ATEMstd::changeAudioChannelMode(uint16_t channelNumber, uint8_t mode
)  {   // Mode: 0=Off, 1=On, 2=AFV
    setAudioMixerInputMixOption(channelNumber, mode);
}
void ATEMstd::changeAudioChannelVolume(uint16_t channelNumber, uint16_t v
olume) {
    setAudioMixerInputVolume(channelNumber, volume);
    /*
    Based on data from the ATEM switcher, this is an approximation to the
 integer value vs. the dB value:
    dB  +60 added   Number from protocol        Interpolated
    6   66  65381       65381
    3   63  46286       46301,04
    0   60  32768       32789,13
    -3  57  23198       23220,37
    -6  54  16423       16444,03
    -9  51  11627       11645,22
    -20 40  3377        3285,93
```

```
    -30 30  1036        1040,21
    -40 20  328     329,3
    -50 10  104     104,24
    -60 0   33      33

    for (int i=-60; i<=6; i=i+3)  {
       Serial.print(i);
       Serial.print(" dB = ");
       Serial.print(33*pow(1.121898585, i+60));
       Serial.println();
    }


    */
}

void ATEMstd::changeAudioMasterVolume(uint16_t volume)  {
    setAudioMixerMasterVolume(volume);
}
void ATEMstd::sendAudioLevelNumbers(bool enable)    {
    setAudioLevelsEnable(enable);
}

// IMCOMPATIBLE with the similar function in old ATEM library  - here you
 enter the official number of the audio source instead!
void ATEMstd::setAudioLevelReadoutChannel(uint16_t AMLv)    {
    _ATEM_AMLv_channel = AMLv;  // Should check that it's in range 0-12
}



void ATEMstd::setWipeReverseDirection(bool reverse) {
    setTransitionWipeReverse(0,reverse);
}



// SPECIAL AUDIO:

/**
 * Get Audio Mixer Levels; Master Left
 */
long ATEMstd::getAudioMixerLevelsMasterLeft() {
    return atemAudioMixerLevelsMasterLeft;
}

/**
 * Get Audio Mixer Levels; Master Right
 */
```

```
long ATEMstd::getAudioMixerLevelsMasterRight() {
    return atemAudioMixerLevelsMasterRight;
}

/**
 * Get Audio Mixer Levels; Monitor
 */
long ATEMstd::getAudioMixerLevelsMonitor() {
    return atemAudioMixerLevelsMonitor;
}

/**
 * Get Audio Mixer Levels; Source Left
 */
long ATEMstd::getAudioMixerLevelsSourceLeft() {
    return atemAudioMixerLevelsSourceLeft;
}

/**
 * Get Audio Mixer Levels; Source Right
 */
long ATEMstd::getAudioMixerLevelsSourceRight() {
    return atemAudioMixerLevelsSourceRight;
}



        // *********************************
        // **
        // ** Implementations in ATEMstd.c:
        // **
        // *********************************

        void ATEMstd::_parseGetCommands(const char *cmdStr) {
            uint8_t mE,keyer,colorGenerator,aUXChannel,mediaPlayer,macroI
ndex;
            uint16_t index,audioSource,sources;
            long temp;
            uint8_t readBytesForTlSr;

            if (!strcmp_P(cmdStr, PSTR("AMLv")))    {
                _readToPacketBuffer(36);
            } else if (!strcmp_P(cmdStr, PSTR("TlSr"))) {
                readBytesForTlSr = ((ATEM_packetBufferLength-2)/3)*3+2;
                _readToPacketBuffer(readBytesForTlSr);
            } else {
                _readToPacketBuffer();  // Default
            }
```

```
            if (!strcmp_P(cmdStr, PSTR("_pin")))    {
                if (_packetBuffer[5]=='T')  {
                        _ATEMmodel = 0;
                } else
                if (_packetBuffer[5]=='1')  {
                        _ATEMmodel = _packetBuffer[29]=='4' ? 4 : 1;
                } else
                if (_packetBuffer[5]=='2')  {
                    _ATEMmodel = _packetBuffer[29]=='4' ? 5 : 2;
                } else
                if (_packetBuffer[5]=='P')  {
                        _ATEMmodel = 3;
                }

                #if ATEM_debug
                if (_serialOutput>0)    {
                    Serial.print(F("Switcher type: "));
                    Serial.print(_ATEMmodel);
                    switch(_ATEMmodel)  {
                        case 0:
                            Serial.println(F(" - TeleVision Studio"));
                        break;
                        case 1:
                            Serial.println(F(" - ATEM 1 M/E"));
                        break;
                        case 2:
                            Serial.println(F(" - ATEM 2 M/E"));
                        break;
                        case 3:
                            Serial.println(F(" - ATEM Production Studio 4
K"));

                        break;
                        case 4:
                            Serial.println(F(" - ATEM 1 M/E 4K"));
                        break;
                        case 5:
                            Serial.println(F(" - ATEM 2 M/E 4K"));
                        break;
                    }
                }
                #endif
            }


            if(!strcmp_P(cmdStr, PSTR("AMLv"))) {
                sources = word(_packetBuffer[0],_packetBuffer[1]);
                if (sources<=24) {
                        atemAudioMixerLevelsMasterLeft = (uint16_t)_packe
tBuffer[5]<<8 | _packetBuffer[6];
```

```
                        atemAudioMixerLevelsMasterRight = (uint16_t)_pack
etBuffer[9]<<8 | _packetBuffer[10];
                        atemAudioMixerLevelsMonitor = (uint16_t)_packetBu
ffer[21]<<8 | _packetBuffer[22];

                        _readToPacketBuffer(sources*2);
                        for(uint8_t a=0;a<sources;a++)  {
                            if (_ATEM_AMLv_channel == word(_packetBuffer[
a<<1], _packetBuffer[(a<<1)+1]))   {

                                if (sources&B1) {   // We must read 4-
byte chunks, so compensate if sources was an odd number
                                    _readToPacketBuffer(2);
                                }

                                for(uint8_t b=0;b<sources;b++)  {
                                    _readToPacketBuffer(16);

                                    if(b==a)    {
                                        atemAudioMixerLevelsSourceLeft =
(uint16_t)_packetBuffer[1]<<8 | _packetBuffer[2];
                                        atemAudioMixerLevelsSourceRight =
 (uint16_t)_packetBuffer[5]<<8 | _packetBuffer[6];
                                        break;
                                    }
                                }
                                break;
                            }
                        }
                }
            }




            if(!strcmp_P(cmdStr, PSTR("_ver"))) {

                    #if ATEM_debug
                    temp = atemProtocolVersionMajor;
                    #endif
                    atemProtocolVersionMajor = word(_packetBuffer[0], _pa
cketBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemProtocolVersionMajor!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemProtocolVersionMajor = "));
                        Serial.println(atemProtocolVersionMajor);
                    }
```

```
                    #endif

                    #if ATEM_debug
                    temp = atemProtocolVersionMinor;
                    #endif
                    atemProtocolVersionMinor = word(_packetBuffer[2], _pa
cketBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemProtocolVersionMinor!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemProtocolVersionMinor = "));
                        Serial.println(atemProtocolVersionMinor);
                    }
                    #endif

            } else
            if(!strcmp_P(cmdStr, PSTR("VidM"))) {

                    #if ATEM_debug
                    temp = atemVideoModeFormat;
                    #endif
                    atemVideoModeFormat = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemVideoModeFormat!=temp
) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemVideoModeFormat = "));
                        Serial.println(atemVideoModeFormat);
                    }
                    #endif

            } else
            if(!strcmp_P(cmdStr, PSTR("PrgI"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemProgramInputVideoSource[mE];
                    #endif
                    atemProgramInputVideoSource[mE] = word(_packetBuffer[
2], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemProgramInputVideoSour
ce[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemProgramInputVideoSource[mE=")
); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemProgramInputVideoSource[mE]);
                    }
                    #endif
```

```
                }
            } else
            if(!strcmp_P(cmdStr, PSTR("PrvI"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemPreviewInputVideoSource[mE];
                    #endif
                    atemPreviewInputVideoSource[mE] = word(_packetBuffer[
2], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemPreviewInputVideoSour
ce[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemPreviewInputVideoSource[mE=")
); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemPreviewInputVideoSource[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("TrSS"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionStyle[mE];
                    #endif
                    atemTransitionStyle[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionStyle[mE]!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemTransitionStyle[mE=")); Seria
l.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionStyle[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionNextTransition[mE];
                    #endif
                    atemTransitionNextTransition[mE] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionNextTransit
ion[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTransitionNextTransition[mE="
)); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionNextTransition[mE]);
```

```
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("TrPr"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionPreviewEnabled[mE];
                    #endif
                    atemTransitionPreviewEnabled[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionPreviewEnab
led[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTransitionPreviewEnabled[mE="
)); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionPreviewEnabled[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("TrPs"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionInTransition[mE];
                    #endif
                    atemTransitionInTransition[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionInTransitio
n[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionInTransition[mE="))
; Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionInTransition[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionFramesRemaining[mE];
                    #endif
                    atemTransitionFramesRemaining[mE] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionFramesRemai
ning[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
```

```
                        Serial.print(F("atemTransitionFramesRemaining[mE=
")); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionFramesRemaining[mE])
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionPosition[mE];
                    #endif
                    atemTransitionPosition[mE] = word(_packetBuffer[4], _
packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionPosition[mE
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionPosition[mE=")); Se
rial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionPosition[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("TMxP"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionMixRate[mE];
                    #endif
                    atemTransitionMixRate[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionMixRate[mE]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionMixRate[mE=")); Ser
ial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionMixRate[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("KeOn"))) {

                mE = _packetBuffer[0];
                keyer = _packetBuffer[1];
                if (mE<=1 && keyer<=3) {
                    #if ATEM_debug
                    temp = atemKeyerOnAirEnabled[mE][keyer];
```

```
                    #endif
                    atemKeyerOnAirEnabled[mE][keyer] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerOnAirEnabled[mE]
[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyerOnAirEnabled[mE=")); Ser
ial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.p
rint(F("] = "));
                        Serial.println(atemKeyerOnAirEnabled[mE][keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("DskP"))) {

                keyer = _packetBuffer[0];
                if (keyer<=1) {
                    #if ATEM_debug
                    temp = atemDownstreamKeyerTie[keyer];
                    #endif
                    atemDownstreamKeyerTie[keyer] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerTie[ke
yer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemDownstreamKeyerTie[keyer="));
 Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerTie[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerRate[keyer];
                    #endif
                    atemDownstreamKeyerRate[keyer] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerRate[k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerRate[keyer="))
; Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerRate[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerPreMultiplied[keyer];
                    #endif
                    atemDownstreamKeyerPreMultiplied[keyer] = _packetBuff
er[3];
```

```
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerPreMultiplied[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemDownstreamKeyerPreMultiplied[keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerPreMultiplied[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerClip[keyer];
                    #endif
                    atemDownstreamKeyerClip[keyer] = word(_packetBuffer[4], _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerClip[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerClip[keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerClip[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerGain[keyer];
                    #endif
                    atemDownstreamKeyerGain[keyer] = word(_packetBuffer[6], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerGain[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerGain[keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerGain[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerInvertKey[keyer];
                    #endif
                    atemDownstreamKeyerInvertKey[keyer] = _packetBuffer[8];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerInvertKey[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemDownstreamKeyerInvertKey[keyer=")); Serial.print(keyer); Serial.print(F("] = "));
```

```
                    Serial.println(atemDownstreamKeyerInvertKey[keyer
]);
                }
                #endif

                #if ATEM_debug
                temp = atemDownstreamKeyerMasked[keyer];
                #endif
                atemDownstreamKeyerMasked[keyer] = _packetBuffer[9];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemDownstreamKeyerMasked
[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                    Serial.print(F("atemDownstreamKeyerMasked[keyer="
)); Serial.print(keyer); Serial.print(F("] = "));
                    Serial.println(atemDownstreamKeyerMasked[keyer]);
                }
                #endif

                #if ATEM_debug
                temp = atemDownstreamKeyerTop[keyer];
                #endif
                atemDownstreamKeyerTop[keyer] = (int16_t) word(_packe
tBuffer[10], _packetBuffer[11]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemDownstreamKeyerTop[ke
yer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                    Serial.print(F("atemDownstreamKeyerTop[keyer="));
 Serial.print(keyer); Serial.print(F("] = "));
                    Serial.println(atemDownstreamKeyerTop[keyer]);
                }
                #endif

                #if ATEM_debug
                temp = atemDownstreamKeyerBottom[keyer];
                #endif
                atemDownstreamKeyerBottom[keyer] = (int16_t) word(_pa
cketBuffer[12], _packetBuffer[13]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemDownstreamKeyerBottom
[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                    Serial.print(F("atemDownstreamKeyerBottom[keyer="
)); Serial.print(keyer); Serial.print(F("] = "));
                    Serial.println(atemDownstreamKeyerBottom[keyer]);
                }
                #endif

                #if ATEM_debug
                temp = atemDownstreamKeyerLeft[keyer];
                #endif
```

```
                    atemDownstreamKeyerLeft[keyer] = (int16_t) word(_pack
etBuffer[14], _packetBuffer[15]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerLeft[k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerLeft[keyer="))
; Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerLeft[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerRight[keyer];
                    #endif
                    atemDownstreamKeyerRight[keyer] = (int16_t) word(_pac
ketBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerRight[
keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerRight[keyer=")
); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerRight[keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("DskS"))) {

                keyer = _packetBuffer[0];
                if (keyer<=1) {
                    #if ATEM_debug
                    temp = atemDownstreamKeyerOnAir[keyer];
                    #endif
                    atemDownstreamKeyerOnAir[keyer] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerOnAir[
keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerOnAir[keyer=")
); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerOnAir[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerInTransition[keyer];
                    #endif
                    atemDownstreamKeyerInTransition[keyer] = _packetBuffe
r[2];
```

```
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerInTran
sition[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerInTransition[k
eyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerInTransition[ke
yer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerIsAutoTransitioning[keyer];
                    #endif
                    atemDownstreamKeyerIsAutoTransitioning[keyer] = _pack
etBuffer[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerIsAuto
Transitioning[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())
) {
                        Serial.print(F("atemDownstreamKeyerIsAutoTransiti
oning[keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerIsAutoTransitio
ning[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerFramesRemaining[keyer];
                    #endif
                    atemDownstreamKeyerFramesRemaining[keyer] = _packetBu
ffer[4];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerFrames
Remaining[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemDownstreamKeyerFramesRemainin
g[keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerFramesRemaining
[keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("FtbP"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemFadeToBlackRate[mE];
```

```
          #endif
          atemFadeToBlackRate[mE] = _packetBuffer[1];
          #if ATEM_debug
          if ((_serialOutput==0x80 && atemFadeToBlackRate[mE]!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
            Serial.print(F("atemFadeToBlackRate[mE=")); Seria
l.print(mE); Serial.print(F("] = "));
            Serial.println(atemFadeToBlackRate[mE]);
          }
          #endif

        }
      } else
      if(!strcmp_P(cmdStr, PSTR("FtbS"))) {

        mE = _packetBuffer[0];
        if (mE<=1) {
          #if ATEM_debug
          temp = atemFadeToBlackStateFullyBlack[mE];
          #endif
          atemFadeToBlackStateFullyBlack[mE] = _packetBuffer[1]
;
          #if ATEM_debug
          if ((_serialOutput==0x80 && atemFadeToBlackStateFully
Black[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
            Serial.print(F("atemFadeToBlackStateFullyBlack[mE
=")); Serial.print(mE); Serial.print(F("] = "));
            Serial.println(atemFadeToBlackStateFullyBlack[mE]
);
          }
          #endif

          #if ATEM_debug
          temp = atemFadeToBlackStateInTransition[mE];
          #endif
          atemFadeToBlackStateInTransition[mE] = _packetBuffer[
2];
          #if ATEM_debug
          if ((_serialOutput==0x80 && atemFadeToBlackStateInTra
nsition[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
            Serial.print(F("atemFadeToBlackStateInTransition[
mE=")); Serial.print(mE); Serial.print(F("] = "));
            Serial.println(atemFadeToBlackStateInTransition[m
E]);
          }
          #endif

          #if ATEM_debug
          temp = atemFadeToBlackStateFramesRemaining[mE];
```

```
                    #endif
                    atemFadeToBlackStateFramesRemaining[mE] = _packetBuff
er[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemFadeToBlackStateFrame
sRemaining[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemFadeToBlackStateFramesRemaini
ng[mE=")); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemFadeToBlackStateFramesRemainin
g[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("AuxS"))) {

                aUXChannel = _packetBuffer[0];
                if (aUXChannel<=5) {
                    #if ATEM_debug
                    temp = atemAuxSourceInput[aUXChannel];
                    #endif
                    atemAuxSourceInput[aUXChannel] = word(_packetBuffer[2
], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAuxSourceInput[aUXCha
nnel]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemAuxSourceInput[aUXChannel="))
; Serial.print(aUXChannel); Serial.print(F("] = "));
                        Serial.println(atemAuxSourceInput[aUXChannel]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("MPCE"))) {

                mediaPlayer = _packetBuffer[0];
                if (mediaPlayer<=1) {
                    #if ATEM_debug
                    temp = atemMediaPlayerSourceType[mediaPlayer];
                    #endif
                    atemMediaPlayerSourceType[mediaPlayer] = _packetBuffe
r[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerSourceType
[mediaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMediaPlayerSourceType[mediaPl
ayer=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
```

```
                        Serial.println(atemMediaPlayerSourceType[mediaPla
yer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemMediaPlayerSourceStillIndex[mediaPlayer];
                    #endif
                    atemMediaPlayerSourceStillIndex[mediaPlayer] = _packe
tBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerSourceStil
lIndex[mediaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemMediaPlayerSourceStillIndex[m
ediaPlayer=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerSourceStillIndex[me
diaPlayer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemMediaPlayerSourceClipIndex[mediaPlayer];
                    #endif
                    atemMediaPlayerSourceClipIndex[mediaPlayer] = _packet
Buffer[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerSourceClip
Index[mediaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemMediaPlayerSourceClipIndex[me
diaPlayer=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerSourceClipIndex[med
iaPlayer]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("MRPr"))) {

                    #if ATEM_debug
                    temp = atemMacroRunStatusState;
                    #endif
                    atemMacroRunStatusState = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMacroRunStatusState!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemMacroRunStatusState = "));
```

```
                        Serial.println(atemMacroRunStatusState);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemMacroRunStatusIsLooping;
                    #endif
                    atemMacroRunStatusIsLooping = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMacroRunStatusIsLoopi
ng!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemMacroRunStatusIsLooping = "))
;
                        Serial.println(atemMacroRunStatusIsLooping);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemMacroRunStatusIndex;
                    #endif
                    atemMacroRunStatusIndex = word(_packetBuffer[2], _pac
ketBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMacroRunStatusIndex!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemMacroRunStatusIndex = "));
                        Serial.println(atemMacroRunStatusIndex);
                    }
                    #endif

            } else
            if(!strcmp_P(cmdStr, PSTR("MPrp"))) {

                macroIndex = _packetBuffer[1];
                if (macroIndex<=9) {
                    #if ATEM_debug
                    temp = atemMacroPropertiesIsUsed[macroIndex];
                    #endif
                    atemMacroPropertiesIsUsed[macroIndex] = _packetBuffer
[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMacroPropertiesIsUsed
[macroIndex]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemMacroPropertiesIsUsed[macroIn
dex=")); Serial.print(macroIndex); Serial.print(F("] = "));
                        Serial.println(atemMacroPropertiesIsUsed[macroInd
ex]);
                    }
                    #endif
```

```
                    memset(atemMacroPropertiesName[macroIndex],0,11);
                    strncpy(atemMacroPropertiesName[macroIndex], (char *)
(_packetBuffer+8), _packetBuffer[5] > 10 ? 10 : _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && hasInitialized()) || (_se
rialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMacroPropertiesName[macroInde
x=")); Serial.print(macroIndex); Serial.print(F("] = "));
                        Serial.println(atemMacroPropertiesName[macroIndex
]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("MRcS"))) {

                    #if ATEM_debug
                    temp = atemMacroRecordingStatusIsRecording;
                    #endif
                    atemMacroRecordingStatusIsRecording = _packetBuffer[0
];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMacroRecordingStatusI
sRecording!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemMacroRecordingStatusIsRecordi
ng = "));

                        Serial.println(atemMacroRecordingStatusIsRecordin
g);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemMacroRecordingStatusIndex;
                    #endif
                    atemMacroRecordingStatusIndex = word(_packetBuffer[2]
, _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMacroRecordingStatusI
ndex!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemMacroRecordingStatusIndex = "
));

                        Serial.println(atemMacroRecordingStatusIndex);
                    }
                    #endif

            } else
            if(!strcmp_P(cmdStr, PSTR("AMIP"))) {
```

```
                audioSource = word(_packetBuffer[0],_packetBuffer[1]);
                if (getAudioSrcIndex(audioSource)<=24) {
                    #if ATEM_debug
                    temp = atemAudioMixerInputMixOption[getAudioSrcIndex(
audioSource)];
                    #endif
                    atemAudioMixerInputMixOption[getAudioSrcIndex(audioSo
urce)] = _packetBuffer[8];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputMixOpt
ion[getAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 && !has
Initialized()))   {
                        Serial.print(F("atemAudioMixerInputMixOption[getA
udioSrcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioSource)
); Serial.print(F("] = "));
                        Serial.println(atemAudioMixerInputMixOption[getAu
dioSrcIndex(audioSource)]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerInputVolume[getAudioSrcIndex(aud
ioSource)];
                    #endif
                    atemAudioMixerInputVolume[getAudioSrcIndex(audioSourc
e)] = word(_packetBuffer[10], _packetBuffer[11]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputVolume
[getAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 && !hasIni
tialized()))  {
                        Serial.print(F("atemAudioMixerInputVolume[getAudi
oSrcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioSource));
Serial.print(F("] = "));
                        Serial.println(atemAudioMixerInputVolume[getAudio
SrcIndex(audioSource)]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerInputBalance[getAudioSrcIndex(au
dioSource)];
                    #endif
                    atemAudioMixerInputBalance[getAudioSrcIndex(audioSour
ce)] = (int16_t) word(_packetBuffer[12], _packetBuffer[13]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputBalanc
e[getAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 && !hasIn
itialized())) {
```

```
                        Serial.print(F("atemAudioMixerInputBalance[getAud
ioSrcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioSource));
 Serial.print(F("] = "));
                        Serial.println(atemAudioMixerInputBalance[getAudi
oSrcIndex(audioSource)]);
                    }
                    #endif

                }
            } else
            if(!strcmp_P(cmdStr, PSTR("TlIn"))) {

                sources = word(_packetBuffer[0],_packetBuffer[1]);
                if (sources<=20) {
                    #if ATEM_debug
                    temp = atemTallyByIndexSources;
                    #endif
                    atemTallyByIndexSources = word(_packetBuffer[0], _pac
ketBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTallyByIndexSources!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemTallyByIndexSources = "));
                        Serial.println(atemTallyByIndexSources);
                    }
                    #endif

                    for(uint8_t a=0;a<sources;a++)  {
                        #if ATEM_debug
                        temp = atemTallyByIndexTallyFlags[a];
                        #endif
                        atemTallyByIndexTallyFlags[a] = _packetBuffer[2+a
];

                        #if ATEM_debug
                        if ((_serialOutput==0x80 && atemTallyByIndexTally
Flags[a]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                            Serial.print(F("atemTallyByIndexTallyFlags[a=
")); Serial.print(a); Serial.print(F("] = "));
                            Serial.println(atemTallyByIndexTallyFlags[a])
;
                        }
                        #endif
                    }

                }
            } else
            {}
        }
```

```
/**
 * Get Protocol Version; Major
 */
uint16_t ATEMstd::getProtocolVersionMajor() {
    return atemProtocolVersionMajor;
}

/**
 * Get Protocol Version; Minor
 */
uint16_t ATEMstd::getProtocolVersionMinor() {
    return atemProtocolVersionMinor;
}

/**
 * Get Video Mode; Format
 */
uint8_t ATEMstd::getVideoModeFormat() {
    return atemVideoModeFormat;
}

/**
 * Set Video Mode; Format
 * format   0: 525i59.94 NTSC, 1: 625i 50 PAL, 2: 525i59.94 N
TSC 16:9, 3: 625i 50 PAL 16:9, 4: 720p50, 5: 720p59.94, 6: 1080i50, 7: 10
80i59.94, 8: 1080p23.98, 9: 1080p24, 10: 1080p25, 11: 1080p29.97, 12: 108
0p50, 13: 1080p59.94, 14: 2160p23.98, 15: 2160p24, 16: 2160p25, 17: 2160p
29.97
 */
void ATEMstd::setVideoModeFormat(uint8_t format) {

    _prepareCommandPacket(PSTR("CVdM"),4);

    _packetBuffer[12+_cBBO+4+4+0] = format;

    _finishCommandPacket();

}

/**
 * Get Program Input; Video Source
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMstd::getProgramInputVideoSource(uint8_t mE) {
```

```
		return atemProgramInputVideoSource[mE];
	}

	/**
	 * Set Program Input; Video Source
	 * mE	0: ME1, 1: ME2
	 * videoSource	(See video source list)
	 */
	void ATEMstd::setProgramInputVideoSource(uint8_t mE, uint16_t
 videoSource) {

		_prepareCommandPacket(PSTR("CPgI"),4,(_packetBuffer[12+_c
BBO+4+4+0]==mE));

		_packetBuffer[12+_cBBO+4+4+0] = mE;

		_packetBuffer[12+_cBBO+4+4+2] = highByte(videoSource);
		_packetBuffer[12+_cBBO+4+4+3] = lowByte(videoSource);

		_finishCommandPacket();

	}

	/**
	 * Get Preview Input; Video Source
	 * mE	0: ME1, 1: ME2
	 */
	uint16_t ATEMstd::getPreviewInputVideoSource(uint8_t mE) {
		return atemPreviewInputVideoSource[mE];
	}

	/**
	 * Set Preview Input; Video Source
	 * mE	0: ME1, 1: ME2
	 * videoSource	(See video source list)
	 */
	void ATEMstd::setPreviewInputVideoSource(uint8_t mE, uint16_t
 videoSource) {

		_prepareCommandPacket(PSTR("CPvI"),4,(_packetBuffer[12+_c
BBO+4+4+0]==mE));

		_packetBuffer[12+_cBBO+4+4+0] = mE;

		_packetBuffer[12+_cBBO+4+4+2] = highByte(videoSource);
		_packetBuffer[12+_cBBO+4+4+3] = lowByte(videoSource);

		_finishCommandPacket();
```

```
}

/**
 * Set Cut; M/E
 * mE   0: ME1, 1: ME2
 */
void ATEMstd::performCutME(uint8_t mE) {

    _prepareCommandPacket(PSTR("DCut"),4);

    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _finishCommandPacket();

}

/**
 * Set Auto; M/E
 * mE   0: ME1, 1: ME2
 */
void ATEMstd::performAutoME(uint8_t mE) {

    _prepareCommandPacket(PSTR("DAut"),4);

    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _finishCommandPacket();

}

/**
 * Get Transition; Style
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMstd::getTransitionStyle(uint8_t mE) {
    return atemTransitionStyle[mE];
}

/**
 * Get Transition; Next Transition
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMstd::getTransitionNextTransition(uint8_t mE) {
    return atemTransitionNextTransition[mE];
}

/**
 * Set Transition; Style
 * mE   0: ME1, 1: ME2
```

```
             * style    0: Mix, 1: Dip, 2: Wipe, 3: DVE, 4: Sting
             */
            void ATEMstd::setTransitionStyle(uint8_t mE, uint8_t style) {

                _prepareCommandPacket(PSTR("CTTp"),4,(_packetBuffer[12+_c
BBO+4+4+1]==mE));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = style;

                _finishCommandPacket();

            }

            /**
             * Set Transition; Next Transition
             * mE   0: ME1, 1: ME2
             * nextTransition   Bit 0: Background=On/Off, Bit 1: Key 1=On
/Off, Bit 2: Key 2=On/Off, Bit 3: Key 3=On/Off, Bit 4: Key 4=On/Off
             */
            void ATEMstd::setTransitionNextTransition(uint8_t mE, uint8_t
 nextTransition) {

                _prepareCommandPacket(PSTR("CTTp"),4,(_packetBuffer[12+_c
BBO+4+4+1]==mE));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+3] = nextTransition;

                _finishCommandPacket();

            }

            /**
             * Get Transition Preview; Enabled
             * mE   0: ME1, 1: ME2
             */
            bool ATEMstd::getTransitionPreviewEnabled(uint8_t mE) {
                return atemTransitionPreviewEnabled[mE];
            }
```

```
/**
 * Set Transition Preview; Enabled
 * mE   0: ME1, 1: ME2
 * enabled  Bit 0: On/Off
 */
void ATEMstd::setTransitionPreviewEnabled(uint8_t mE, bool en
abled) {

    _prepareCommandPacket(PSTR("CTPr"),4,(_packetBuffer[12+_c
BBO+4+4+0]==mE));

    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _packetBuffer[12+_cBBO+4+4+1] = enabled;

    _finishCommandPacket();

}

/**
 * Get Transition Position; In Transition
 * mE   0: ME1, 1: ME2
 */
bool ATEMstd::getTransitionInTransition(uint8_t mE) {
    return atemTransitionInTransition[mE];
}

/**
 * Get Transition Position; Frames Remaining
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMstd::getTransitionFramesRemaining(uint8_t mE) {
    return atemTransitionFramesRemaining[mE];
}

/**
 * Get Transition Position; Position
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMstd::getTransitionPosition(uint8_t mE) {
    return atemTransitionPosition[mE];
}

/**
 * Set Transition Position; Position
 * mE   0: ME1, 1: ME2
 * position     0-9999
 */
```

```
            void ATEMstd::setTransitionPosition(uint8_t mE, uint16_t posi
tion) {

                _prepareCommandPacket(PSTR("CTPs"),4,(_packetBuffer[12+_c
BBO+4+4+0]==mE));

                _packetBuffer[12+_cBBO+4+4+0] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(position);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(position);

                _finishCommandPacket();

            }

            /**
             * Get Transition Mix; Rate
             * mE    0: ME1, 1: ME2
             */
            uint8_t ATEMstd::getTransitionMixRate(uint8_t mE) {
                return atemTransitionMixRate[mE];
            }

            /**
             * Set Transition Mix; Rate
             * mE    0: ME1, 1: ME2
             * rate      1-250: Frames
             */
            void ATEMstd::setTransitionMixRate(uint8_t mE, uint8_t rate)
{

                _prepareCommandPacket(PSTR("CTMx"),4,(_packetBuffer[12+_c
BBO+4+4+0]==mE));

                _packetBuffer[12+_cBBO+4+4+0] = mE;

                _packetBuffer[12+_cBBO+4+4+1] = rate;

                _finishCommandPacket();

            }

            /**
             * Set Transition Wipe; Rate
             * mE    0: ME1, 1: ME2
             * rate      1-250: Frames
             */
            void ATEMstd::setTransitionWipeRate(uint8_t mE, uint8_t rate)
 {
```

```
        _prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

            // Set Mask: 1
        _packetBuffer[12+_cBBO+4+4+1] |= 1;

        _packetBuffer[12+_cBBO+4+4+2] = mE;

        _packetBuffer[12+_cBBO+4+4+3] = rate;

        _finishCommandPacket();

    }

    /**
     * Set Transition Wipe; Pattern
     * mE    0: ME1, 1: ME2
     * pattern  0-17: Pattern Styles
     */
    void ATEMstd::setTransitionWipePattern(uint8_t mE, uint8_t pa
ttern) {

        _prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

            // Set Mask: 2
        _packetBuffer[12+_cBBO+4+4+1] |= 2;

        _packetBuffer[12+_cBBO+4+4+2] = mE;

        _packetBuffer[12+_cBBO+4+4+4] = pattern;

        _finishCommandPacket();

    }

    /**
     * Set Transition Wipe; Width
     * mE    0: ME1, 1: ME2
     * width    0-10000: 0-100%
     */
    void ATEMstd::setTransitionWipeWidth(uint8_t mE, uint16_t wid
th) {

        _prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

            // Set Mask: 4
```

```
				_packetBuffer[12+_cBBO+4+4+1] |= 4;

				_packetBuffer[12+_cBBO+4+4+2] = mE;

				_packetBuffer[12+_cBBO+4+4+6] = highByte(width);
				_packetBuffer[12+_cBBO+4+4+7] = lowByte(width);

				_finishCommandPacket();

			}

			/**
			 * Set Transition Wipe; Fill Source
			 * mE	0: ME1, 1: ME2
			 * fillSource	(See video source list)
			 */
			void ATEMstd::setTransitionWipeFillSource(uint8_t mE, uint16_
t fillSource) {

				_prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

					// Set Mask: 8
				_packetBuffer[12+_cBBO+4+4+1] |= 8;

				_packetBuffer[12+_cBBO+4+4+2] = mE;

				_packetBuffer[12+_cBBO+4+4+8] = highByte(fillSource);
				_packetBuffer[12+_cBBO+4+4+9] = lowByte(fillSource);

				_finishCommandPacket();

			}

			/**
			 * Set Transition Wipe; Symmetry
			 * mE	0: ME1, 1: ME2
			 * symmetry	0-10000: 0-100%
			 */
			void ATEMstd::setTransitionWipeSymmetry(uint8_t mE, uint16_t
symmetry) {

				_prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

					// Set Mask: 16
				_packetBuffer[12+_cBBO+4+4+1] |= 16;

				_packetBuffer[12+_cBBO+4+4+2] = mE;
```

```
        _packetBuffer[12+_cBBO+4+4+10] = highByte(symmetry);
        _packetBuffer[12+_cBBO+4+4+11] = lowByte(symmetry);

        _finishCommandPacket();

    }

    /**
     * Set Transition Wipe; Softness
     * mE    0: ME1, 1: ME2
     * softness     0-10000: 0-100%
     */
    void ATEMstd::setTransitionWipeSoftness(uint8_t mE, uint16_t
softness) {

        _prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

            // Set Mask: 32
        _packetBuffer[12+_cBBO+4+4+1] |= 32;

        _packetBuffer[12+_cBBO+4+4+2] = mE;

        _packetBuffer[12+_cBBO+4+4+12] = highByte(softness);
        _packetBuffer[12+_cBBO+4+4+13] = lowByte(softness);

        _finishCommandPacket();

    }

    /**
     * Set Transition Wipe; Position X
     * mE    0: ME1, 1: ME2
     * positionX    0-10000: 0.0-1.0
     */
    void ATEMstd::setTransitionWipePositionX(uint8_t mE, uint16_t
 positionX) {

        _prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

            // Set Mask: 64
        _packetBuffer[12+_cBBO+4+4+1] |= 64;

        _packetBuffer[12+_cBBO+4+4+2] = mE;

        _packetBuffer[12+_cBBO+4+4+14] = highByte(positionX);
        _packetBuffer[12+_cBBO+4+4+15] = lowByte(positionX);
```

```
                _finishCommandPacket();

            }

            /**
             * Set Transition Wipe; Position Y
             * mE   0: ME1, 1: ME2
             * positionY    0-10000: 0.0-1.0
             */
            void ATEMstd::setTransitionWipePositionY(uint8_t mE, uint16_t
 positionY) {

                _prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

                    // Set Mask: 128
                _packetBuffer[12+_cBBO+4+4+1] |= 128;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(positionY);
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(positionY);

                _finishCommandPacket();

            }

            /**
             * Set Transition Wipe; Reverse
             * mE   0: ME1, 1: ME2
             * reverse  Bit 0: On/Off
             */
            void ATEMstd::setTransitionWipeReverse(uint8_t mE, bool rever
se) {

                _prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

                    // Set Mask: 256
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+18] = reverse;

                _finishCommandPacket();

            }
```

```
		/**
		 * Set Transition Wipe; FlipFlop
		 * mE	0: ME1, 1: ME2
		 * flipFlop	Bit 0: On/Off
		 */
		void ATEMstd::setTransitionWipeFlipFlop(uint8_t mE, bool flip
Flop) {

			_prepareCommandPacket(PSTR("CTWp"),20,(_packetBuffer[12+_
cBBO+4+4+2]==mE));

				// Set Mask: 512
			_packetBuffer[12+_cBBO+4+4+0] |= 2;

			_packetBuffer[12+_cBBO+4+4+2] = mE;

			_packetBuffer[12+_cBBO+4+4+19] = flipFlop;

			_finishCommandPacket();

		}

		/**
		 * Get Keyer On Air; Enabled
		 * mE	0: ME1, 1: ME2
		 * keyer	0-3: Keyer 1-4
		 */
		bool ATEMstd::getKeyerOnAirEnabled(uint8_t mE, uint8_t keyer)
 {
			return atemKeyerOnAirEnabled[mE][keyer];
		}

		/**
		 * Set Keyer On Air; Enabled
		 * mE	0: ME1, 1: ME2
		 * keyer	0-3: Keyer 1-4
		 * enabled	Bit 0: On/Off
		 */
		void ATEMstd::setKeyerOnAirEnabled(uint8_t mE, uint8_t keyer,
 bool enabled) {

			_prepareCommandPacket(PSTR("CKOn"),4,(_packetBuffer[12+_c
BBO+4+4+0]==mE) && (_packetBuffer[12+_cBBO+4+4+1]==keyer));

			_packetBuffer[12+_cBBO+4+4+0] = mE;

			_packetBuffer[12+_cBBO+4+4+1] = keyer;
```

```
                _packetBuffer[12+_cBBO+4+4+2] = enabled;

                _finishCommandPacket();

            }

            /**
             * Set Key Mask; Masked
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * masked    Bit 0: On/Off
             */
            void ATEMstd::setKeyerMasked(uint8_t mE, uint8_t keyer, bool
masked) {

                _prepareCommandPacket(PSTR("CKMs"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+3] = masked;

                _finishCommandPacket();

            }

            /**
             * Set Key Mask; Top
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * top   -9000-9000: -9.00-9.00
             */
            void ATEMstd::setKeyerTop(uint8_t mE, uint8_t keyer, int16_t
top) {

                _prepareCommandPacket(PSTR("CKMs"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;
```

```
		_packetBuffer[12+_cBBO+4+4+4] = highByte(top);
		_packetBuffer[12+_cBBO+4+4+5] = lowByte(top);

		_finishCommandPacket();

	}

	/**
	 * Set Key Mask; Bottom
	 * mE	0: ME1, 1: ME2
	 * keyer	0-3: Keyer 1-4
	 * bottom	-9000-9000: -9.00-9.00
	 */
	void ATEMstd::setKeyerBottom(uint8_t mE, uint8_t keyer, int16
_t bottom) {

		_prepareCommandPacket(PSTR("CKMs"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

			// Set Mask: 4
		_packetBuffer[12+_cBBO+4+4+0] |= 4;

		_packetBuffer[12+_cBBO+4+4+1] = mE;

		_packetBuffer[12+_cBBO+4+4+2] = keyer;

		_packetBuffer[12+_cBBO+4+4+6] = highByte(bottom);
		_packetBuffer[12+_cBBO+4+4+7] = lowByte(bottom);

		_finishCommandPacket();

	}

	/**
	 * Set Key Mask; Left
	 * mE	0: ME1, 1: ME2
	 * keyer	0-3: Keyer 1-4
	 * left	-16000-16000: -9.00-9.00
	 */
	void ATEMstd::setKeyerLeft(uint8_t mE, uint8_t keyer, int16_t
 left) {

		_prepareCommandPacket(PSTR("CKMs"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

			// Set Mask: 8
		_packetBuffer[12+_cBBO+4+4+0] |= 8;
```

```
            _packetBuffer[12+_cBBO+4+4+1] = mE;

            _packetBuffer[12+_cBBO+4+4+2] = keyer;

            _packetBuffer[12+_cBBO+4+4+8] = highByte(left);
            _packetBuffer[12+_cBBO+4+4+9] = lowByte(left);

            _finishCommandPacket();

        }

        /**
         * Set Key Mask; Right
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * right     -16000-16000: -9.00-9.00
         */
        void ATEMstd::setKeyerRight(uint8_t mE, uint8_t keyer, int16_
t right) {

            _prepareCommandPacket(PSTR("CKMs"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                // Set Mask: 16
            _packetBuffer[12+_cBBO+4+4+0] |= 16;

            _packetBuffer[12+_cBBO+4+4+1] = mE;

            _packetBuffer[12+_cBBO+4+4+2] = keyer;

            _packetBuffer[12+_cBBO+4+4+10] = highByte(right);
            _packetBuffer[12+_cBBO+4+4+11] = lowByte(right);

            _finishCommandPacket();

        }

        /**
         * Set Key Fill; Fill Source
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * fillSource    (See video source list)
         */
        void ATEMstd::setKeyerFillSource(uint8_t mE, uint8_t keyer, u
int16_t fillSource) {

            _prepareCommandPacket(PSTR("CKeF"),4,(_packetBuffer[12+_c
BBO+4+4+0]==mE) && (_packetBuffer[12+_cBBO+4+4+1]==keyer));
```

```
                _packetBuffer[12+_cBBO+4+4+0] = mE;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(fillSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(fillSource);

                _finishCommandPacket();

            }

            /**
             * Set Key Luma; Pre Multiplied
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * preMultiplied     Bit 0: On/Off
             */
            void ATEMstd::setKeyLumaPreMultiplied(uint8_t mE, uint8_t key
er, bool preMultiplied) {

                _prepareCommandPacket(PSTR("CKLm"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+3] = preMultiplied;

                _finishCommandPacket();

            }

            /**
             * Set Key Luma; Clip
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * clip      0-1000: 0-100%
             */
            void ATEMstd::setKeyLumaClip(uint8_t mE, uint8_t keyer, uint1
6_t clip) {

                _prepareCommandPacket(PSTR("CKLm"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 2
```

```
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(clip);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(clip);

                _finishCommandPacket();

            }

            /**
             * Set Key Luma; Gain
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * gain      0-1000: 0-100%
             */
            void ATEMstd::setKeyLumaGain(uint8_t mE, uint8_t keyer, uint1
6_t gain) {

                _prepareCommandPacket(PSTR("CKLm"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+0] |= 4;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+6] = highByte(gain);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(gain);

                _finishCommandPacket();

            }

            /**
             * Set Key Luma; Invert Key
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * invertKey     Bit 0: On/Off
             */
            void ATEMstd::setKeyLumaInvertKey(uint8_t mE, uint8_t keyer,
bool invertKey) {
```

```
        _prepareCommandPacket(PSTR("CKLm"),12,(_packetBuffer[12+_
cBBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

            // Set Mask: 8
        _packetBuffer[12+_cBBO+4+4+0] |= 8;

        _packetBuffer[12+_cBBO+4+4+1] = mE;

        _packetBuffer[12+_cBBO+4+4+2] = keyer;

        _packetBuffer[12+_cBBO+4+4+8] = invertKey;

        _finishCommandPacket();

    }

    /**
     * Set Key DVE; Size X
     * mE    0: ME1, 1: ME2
     * keyer     0-3: Keyer 1-4
     * sizeX     Example: 1000: 1.000
     */
    void ATEMstd::setKeyDVESizeX(uint8_t mE, uint8_t keyer, int32
_t sizeX) {

        _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

            // Set Mask: 1
        _packetBuffer[12+_cBBO+4+4+3] |= 1;

        _packetBuffer[12+_cBBO+4+4+4] = mE;

        _packetBuffer[12+_cBBO+4+4+5] = keyer;

        _packetBuffer[12+_cBBO+4+4+8] = (int32_t)((sizeX>>24) & 0
xFF);
        _packetBuffer[12+_cBBO+4+4+9] = (int32_t)((sizeX>>16) & 0
xFF);
        _packetBuffer[12+_cBBO+4+4+10] = (int32_t)((sizeX>>8) & 0
xFF);
        _packetBuffer[12+_cBBO+4+4+11] = (int32_t)(sizeX & 0xFF);

        _finishCommandPacket();

    }

    /**
     * Set Key DVE; Size Y
```

```
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 * sizeY    Example: 2000: 2.000
 */
void ATEMstd::setKeyDVESizeY(uint8_t mE, uint8_t keyer, int32
_t sizeY) {

    _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

        // Set Mask: 2
    _packetBuffer[12+_cBBO+4+4+3] |= 2;

    _packetBuffer[12+_cBBO+4+4+4] = mE;

    _packetBuffer[12+_cBBO+4+4+5] = keyer;

    _packetBuffer[12+_cBBO+4+4+12] = (int32_t)((sizeY>>24) &
0xFF);
    _packetBuffer[12+_cBBO+4+4+13] = (int32_t)((sizeY>>16) &
0xFF);
    _packetBuffer[12+_cBBO+4+4+14] = (int32_t)((sizeY>>8) & 0
xFF);
    _packetBuffer[12+_cBBO+4+4+15] = (int32_t)(sizeY & 0xFF);

    _finishCommandPacket();

}

/**
 * Set Key DVE; Position X
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 * positionX    Example: 1000: 1.000
 */
void ATEMstd::setKeyDVEPositionX(uint8_t mE, uint8_t keyer, i
nt32_t positionX) {

    _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

        // Set Mask: 4
    _packetBuffer[12+_cBBO+4+4+3] |= 4;

    _packetBuffer[12+_cBBO+4+4+4] = mE;

    _packetBuffer[12+_cBBO+4+4+5] = keyer;
```

```
            _packetBuffer[12+_cBBO+4+4+16] = (int32_t)((positionX>>24
) & 0xFF);
            _packetBuffer[12+_cBBO+4+4+17] = (int32_t)((positionX>>16
) & 0xFF);
            _packetBuffer[12+_cBBO+4+4+18] = (int32_t)((positionX>>8)
 & 0xFF);
            _packetBuffer[12+_cBBO+4+4+19] = (int32_t)(positionX & 0x
FF);

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Position Y
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * positionY     Example: -1000: -1.000
         */
        void ATEMstd::setKeyDVEPositionY(uint8_t mE, uint8_t keyer, i
nt32_t positionY) {

            _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 8
            _packetBuffer[12+_cBBO+4+4+3] |= 8;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+20] = (int32_t)((positionY>>24
) & 0xFF);
            _packetBuffer[12+_cBBO+4+4+21] = (int32_t)((positionY>>16
) & 0xFF);
            _packetBuffer[12+_cBBO+4+4+22] = (int32_t)((positionY>>8)
 & 0xFF);
            _packetBuffer[12+_cBBO+4+4+23] = (int32_t)(positionY & 0x
FF);

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Rotation
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
```

```
             * rotation     Example: 3670: 1 rotation+7 degress
             */
            void ATEMstd::setKeyDVERotation(uint8_t mE, uint8_t keyer, in
t32_t rotation) {

                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 16
                _packetBuffer[12+_cBBO+4+4+3] |= 16;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+24] = (int32_t)((rotation>>24)
 & 0xFF);
                _packetBuffer[12+_cBBO+4+4+25] = (int32_t)((rotation>>16)
 & 0xFF);
                _packetBuffer[12+_cBBO+4+4+26] = (int32_t)((rotation>>8)
& 0xFF);
                _packetBuffer[12+_cBBO+4+4+27] = (int32_t)(rotation & 0xF
F);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Enabled
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * borderEnabled    Bit 0: On/Off
             */
            void ATEMstd::setKeyDVEBorderEnabled(uint8_t mE, uint8_t keye
r, bool borderEnabled) {

                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 32
                _packetBuffer[12+_cBBO+4+4+3] |= 32;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+28] = borderEnabled;
```

```
            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Shadow
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * shadow    Bit 0: On/Off
         */
        void ATEMstd::setKeyDVEShadow(uint8_t mE, uint8_t keyer, bool
 shadow) {

            _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 64
            _packetBuffer[12+_cBBO+4+4+3] |= 64;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+29] = shadow;

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Border Bevel
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * borderBevel  0: No, 1: In/Out, 2: In, 3: Out
         */
        void ATEMstd::setKeyDVEBorderBevel(uint8_t mE, uint8_t keyer,
 uint8_t borderBevel) {

            _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 128
            _packetBuffer[12+_cBBO+4+4+3] |= 128;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+30] = borderBevel;
```

```
            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Border Outer Width
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * borderOuterWidth     0-1600: 0-16.00
         */
        void ATEMstd::setKeyDVEBorderOuterWidth(uint8_t mE, uint8_t k
eyer, uint16_t borderOuterWidth) {

            _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 256
            _packetBuffer[12+_cBBO+4+4+2] |= 1;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+32] = highByte(borderOuterWidt
h);
            _packetBuffer[12+_cBBO+4+4+33] = lowByte(borderOuterWidth
);

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Border Inner Width
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * borderInnerWidth     0-1600: 0-16.00
         */
        void ATEMstd::setKeyDVEBorderInnerWidth(uint8_t mE, uint8_t k
eyer, uint16_t borderInnerWidth) {

            _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 512
            _packetBuffer[12+_cBBO+4+4+2] |= 2;

            _packetBuffer[12+_cBBO+4+4+4] = mE;
```

```
			_packetBuffer[12+_cBBO+4+4+5] = keyer;

			_packetBuffer[12+_cBBO+4+4+34] = highByte(borderInnerWidt
h);
			_packetBuffer[12+_cBBO+4+4+35] = lowByte(borderInnerWidth
);

			_finishCommandPacket();

		}

		/**
		 * Set Key DVE; Border Outer Softness
		 * mE	0: ME1, 1: ME2
		 * keyer	0-3: Keyer 1-4
		 * borderOuterSoftness	0-100: 0-100%
		 */
		void ATEMstd::setKeyDVEBorderOuterSoftness(uint8_t mE, uint8_
t keyer, uint8_t borderOuterSoftness) {

			_prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

				// Set Mask: 1024
			_packetBuffer[12+_cBBO+4+4+2] |= 4;

			_packetBuffer[12+_cBBO+4+4+4] = mE;

			_packetBuffer[12+_cBBO+4+4+5] = keyer;

			_packetBuffer[12+_cBBO+4+4+36] = borderOuterSoftness;

			_finishCommandPacket();

		}

		/**
		 * Set Key DVE; Border Inner Softness
		 * mE	0: ME1, 1: ME2
		 * keyer	0-3: Keyer 1-4
		 * borderInnerSoftness	0-100: 0-100%
		 */
		void ATEMstd::setKeyDVEBorderInnerSoftness(uint8_t mE, uint8_
t keyer, uint8_t borderInnerSoftness) {

			_prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));
```

```
                // Set Mask: 2048
            _packetBuffer[12+_cBBO+4+4+2] |= 8;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+37] = borderInnerSoftness;

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Border Bevel Softness
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * borderBevelSoftness  0-100: 0.0-1.0
         */
        void ATEMstd::setKeyDVEBorderBevelSoftness(uint8_t mE, uint8_
t keyer, uint8_t borderBevelSoftness) {

            _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 4096
            _packetBuffer[12+_cBBO+4+4+2] |= 16;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+38] = borderBevelSoftness;

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Border Bevel Position
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * borderBevelPosition  0-100: 0.0-1.0
         */
        void ATEMstd::setKeyDVEBorderBevelPosition(uint8_t mE, uint8_
t keyer, uint8_t borderBevelPosition) {

            _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));
```

```
                // Set Mask: 8192
            _packetBuffer[12+_cBBO+4+4+2] |= 32;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+39] = borderBevelPosition;

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Border Opacity
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * borderOpacity     0-100: 0-100%
         */
        void ATEMstd::setKeyDVEBorderOpacity(uint8_t mE, uint8_t keye
r, uint8_t borderOpacity) {

            _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 16384
            _packetBuffer[12+_cBBO+4+4+2] |= 64;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+40] = borderOpacity;

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Border Hue
         * mE    0: ME1, 1: ME2
         * keyer     0-3: Keyer 1-4
         * borderHue     0-3599: 0-359.9 Degrees
         */
        void ATEMstd::setKeyDVEBorderHue(uint8_t mE, uint8_t keyer, u
int16_t borderHue) {
```

```
                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 32768
                _packetBuffer[12+_cBBO+4+4+2] |= 128;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+42] = highByte(borderHue);
                _packetBuffer[12+_cBBO+4+4+43] = lowByte(borderHue);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Saturation
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * borderSaturation     0-1000: 0-100%
             */
            void ATEMstd::setKeyDVEBorderSaturation(uint8_t mE, uint8_t k
eyer, uint16_t borderSaturation) {

                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 65536
                _packetBuffer[12+_cBBO+4+4+1] |= 1;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+44] = highByte(borderSaturatio
n);
                _packetBuffer[12+_cBBO+4+4+45] = lowByte(borderSaturation
);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Luma
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
```

```
             * borderLuma   0-1000: 0-100%
             */
            void ATEMstd::setKeyDVEBorderLuma(uint8_t mE, uint8_t keyer,
uint16_t borderLuma) {

                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 131072
                _packetBuffer[12+_cBBO+4+4+1] |= 2;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+46] = highByte(borderLuma);
                _packetBuffer[12+_cBBO+4+4+47] = lowByte(borderLuma);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Light Source Direction
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * lightSourceDirection     0-3590: 0-359 Degrees
             */
            void ATEMstd::setKeyDVELightSourceDirection(uint8_t mE, uint8
_t keyer, uint16_t lightSourceDirection) {

                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 262144
                _packetBuffer[12+_cBBO+4+4+1] |= 4;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+48] = highByte(lightSourceDire
ction);
                _packetBuffer[12+_cBBO+4+4+49] = lowByte(lightSourceDirec
tion);

                _finishCommandPacket();

            }
```

```
/**
 * Set Key DVE; Light Source Altitude
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 * lightSourceAltitude  10-100: 10-100
 */
void ATEMstd::setKeyDVELightSourceAltitude(uint8_t mE, uint8_t keyer, uint8_t lightSourceAltitude) {

    _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

        // Set Mask: 524288
    _packetBuffer[12+_cBBO+4+4+1] |= 8;

    _packetBuffer[12+_cBBO+4+4+4] = mE;

    _packetBuffer[12+_cBBO+4+4+5] = keyer;

    _packetBuffer[12+_cBBO+4+4+50] = lightSourceAltitude;

    _finishCommandPacket();

}

/**
 * Set Key DVE; Masked
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 * masked   Bit 0: On/Off
 */
void ATEMstd::setKeyDVEMasked(uint8_t mE, uint8_t keyer, bool masked) {

    _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

        // Set Mask: 1048576
    _packetBuffer[12+_cBBO+4+4+1] |= 16;

    _packetBuffer[12+_cBBO+4+4+4] = mE;

    _packetBuffer[12+_cBBO+4+4+5] = keyer;

    _packetBuffer[12+_cBBO+4+4+51] = masked;

    _finishCommandPacket();
```

```
            }

            /**
             * Set Key DVE; Top
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * top  -9000-9000: -9.00-9.00
             */
            void ATEMstd::setKeyDVETop(uint8_t mE, uint8_t keyer, int16_t
 top) {

                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 2097152
                _packetBuffer[12+_cBBO+4+4+1] |= 32;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+52] = highByte(top);
                _packetBuffer[12+_cBBO+4+4+53] = lowByte(top);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Bottom
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * bottom   -9000-9000: -9.00-9.00
             */
            void ATEMstd::setKeyDVEBottom(uint8_t mE, uint8_t keyer, int1
6_t bottom) {

                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 4194304
                _packetBuffer[12+_cBBO+4+4+1] |= 64;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+54] = highByte(bottom);
                _packetBuffer[12+_cBBO+4+4+55] = lowByte(bottom);
```

```
        _finishCommandPacket();

    }

    /**
     * Set Key DVE; Left
     * mE   0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     * left     -16000-16000: -9.00-9.00
     */
    void ATEMstd::setKeyDVELeft(uint8_t mE, uint8_t keyer, int16_
t left) {

        _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

            // Set Mask: 8388608
        _packetBuffer[12+_cBBO+4+4+1] |= 128;

        _packetBuffer[12+_cBBO+4+4+4] = mE;

        _packetBuffer[12+_cBBO+4+4+5] = keyer;

        _packetBuffer[12+_cBBO+4+4+56] = highByte(left);
        _packetBuffer[12+_cBBO+4+4+57] = lowByte(left);

        _finishCommandPacket();

    }

    /**
     * Set Key DVE; Right
     * mE   0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     * right    -16000-16000: -9.00-9.00
     */
    void ATEMstd::setKeyDVERight(uint8_t mE, uint8_t keyer, int16
_t right) {

        _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

            // Set Mask: 16777216
        _packetBuffer[12+_cBBO+4+4+0] |= 1;

        _packetBuffer[12+_cBBO+4+4+4] = mE;

        _packetBuffer[12+_cBBO+4+4+5] = keyer;
```

```
                _packetBuffer[12+_cBBO+4+4+58] = highByte(right);
                _packetBuffer[12+_cBBO+4+4+59] = lowByte(right);

                _finishCommandPacket();

        }

        /**
         * Set Key DVE; Rate
         * mE   0: ME1, 1: ME2
         * keyer    0-3: Keyer 1-4
         * rate     1-250: Frames
         */
        void ATEMstd::setKeyDVERate(uint8_t mE, uint8_t keyer, uint8_
t rate) {

                _prepareCommandPacket(PSTR("CKDV"),64,(_packetBuffer[12+_
cBBO+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 33554432
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+60] = rate;

                _finishCommandPacket();

        }

        /**
         * Set Run Flying Key; Key Frame
         * mE   0: ME1, 1: ME2
         * keyer    0-3: Keyer 1-4
         * keyFrame     1: A, 2: B, 3: Full, 4: Run-To-Infinite
         */
        void ATEMstd::setRunFlyingKeyKeyFrame(uint8_t mE, uint8_t key
er, uint8_t keyFrame) {

                _prepareCommandPacket(PSTR("RFlK"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;
```

```
                _packetBuffer[12+_cBBO+4+4+4] = keyFrame;

                _finishCommandPacket();

            }

            /**
             * Set Run Flying Key; Run-to-Infinite-index
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * runtoInfiniteindex
             */
            void ATEMstd::setRunFlyingKeyRuntoInfiniteindex(uint8_t mE, u
int8_t keyer, uint8_t runtoInfiniteindex) {

                _prepareCommandPacket(PSTR("RFlK"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+5] = runtoInfiniteindex;

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Fill Source
             * keyer     0-3: Keyer 1-4
             * fillSource    (See video source list)
             */
            void ATEMstd::setDownstreamKeyerFillSource(uint8_t keyer, uin
t16_t fillSource) {

                _prepareCommandPacket(PSTR("CDsF"),4,(_packetBuffer[12+_c
BBO+4+4+0]==keyer));

                _packetBuffer[12+_cBBO+4+4+0] = keyer;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(fillSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(fillSource);

                _finishCommandPacket();
```

```
	}

	/**
	 * Set Downstream Keyer; Key Source
	 * keyer    0-3: Keyer 1-4
	 * keySource    (See video source list)
	 */
	void ATEMstd::setDownstreamKeyerKeySource(uint8_t keyer, uint
16_t keySource) {

		_prepareCommandPacket(PSTR("CDsC"),4,(_packetBuffer[12+_c
BBO+4+4+0]==keyer));

		_packetBuffer[12+_cBBO+4+4+0] = keyer;

		_packetBuffer[12+_cBBO+4+4+2] = highByte(keySource);
		_packetBuffer[12+_cBBO+4+4+3] = lowByte(keySource);

		_finishCommandPacket();

	}

	/**
	 * Get Downstream Keyer; Tie
	 * keyer    0: DSK1, 1: DSK2
	 */
	bool ATEMstd::getDownstreamKeyerTie(uint8_t keyer) {
		return atemDownstreamKeyerTie[keyer];
	}

	/**
	 * Get Downstream Keyer; Rate
	 * keyer    0: DSK1, 1: DSK2
	 */
	uint8_t ATEMstd::getDownstreamKeyerRate(uint8_t keyer) {
		return atemDownstreamKeyerRate[keyer];
	}

	/**
	 * Get Downstream Keyer; Pre Multiplied
	 * keyer    0: DSK1, 1: DSK2
	 */
	bool ATEMstd::getDownstreamKeyerPreMultiplied(uint8_t keyer)
{
		return atemDownstreamKeyerPreMultiplied[keyer];
	}

	/**
	 * Get Downstream Keyer; Clip
```

```
 * keyer    0: DSK1, 1: DSK2
 */
uint16_t ATEMstd::getDownstreamKeyerClip(uint8_t keyer) {
    return atemDownstreamKeyerClip[keyer];
}

/**
 * Get Downstream Keyer; Gain
 * keyer    0: DSK1, 1: DSK2
 */
uint16_t ATEMstd::getDownstreamKeyerGain(uint8_t keyer) {
    return atemDownstreamKeyerGain[keyer];
}

/**
 * Get Downstream Keyer; Invert Key
 * keyer    0: DSK1, 1: DSK2
 */
bool ATEMstd::getDownstreamKeyerInvertKey(uint8_t keyer) {
    return atemDownstreamKeyerInvertKey[keyer];
}

/**
 * Get Downstream Keyer; Masked
 * keyer    0: DSK1, 1: DSK2
 */
bool ATEMstd::getDownstreamKeyerMasked(uint8_t keyer) {
    return atemDownstreamKeyerMasked[keyer];
}

/**
 * Get Downstream Keyer; Top
 * keyer    0: DSK1, 1: DSK2
 */
int16_t ATEMstd::getDownstreamKeyerTop(uint8_t keyer) {
    return atemDownstreamKeyerTop[keyer];
}

/**
 * Get Downstream Keyer; Bottom
 * keyer    0: DSK1, 1: DSK2
 */
int16_t ATEMstd::getDownstreamKeyerBottom(uint8_t keyer) {
    return atemDownstreamKeyerBottom[keyer];
}

/**
 * Get Downstream Keyer; Left
 * keyer    0: DSK1, 1: DSK2
```

```
 */
int16_t ATEMstd::getDownstreamKeyerLeft(uint8_t keyer) {
    return atemDownstreamKeyerLeft[keyer];
}

/**
 * Get Downstream Keyer; Right
 * keyer    0: DSK1, 1: DSK2
 */
int16_t ATEMstd::getDownstreamKeyerRight(uint8_t keyer) {
    return atemDownstreamKeyerRight[keyer];
}

/**
 * Set Downstream Keyer; Tie
 * keyer    0: DSK1, 1: DSK2
 * tie  Bit 0: On/Off
 */
void ATEMstd::setDownstreamKeyerTie(uint8_t keyer, bool tie)
{

    _prepareCommandPacket(PSTR("CDsT"),4,(_packetBuffer[12+_c
BBO+4+4+0]==keyer));

    _packetBuffer[12+_cBBO+4+4+0] = keyer;

    _packetBuffer[12+_cBBO+4+4+1] = tie;

    _finishCommandPacket();

}

/**
 * Set Downstream Keyer; Pre Multiplied
 * keyer    0-3: Keyer 1-4
 * preMultiplied    Bit 0: On/Off
 */
void ATEMstd::setDownstreamKeyerPreMultiplied(uint8_t keyer,
bool preMultiplied) {

    _prepareCommandPacket(PSTR("CDsG"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+1] = keyer;

    _packetBuffer[12+_cBBO+4+4+2] = preMultiplied;
```

```
            _finishCommandPacket();

        }

        /**
         * Set Downstream Keyer; Clip
         * keyer    0-3: Keyer 1-4
         * clip     0-1000: 0-100%
         */
        void ATEMstd::setDownstreamKeyerClip(uint8_t keyer, uint16_t
clip) {

            _prepareCommandPacket(PSTR("CDsG"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

                // Set Mask: 2
            _packetBuffer[12+_cBBO+4+4+0] |= 2;

            _packetBuffer[12+_cBBO+4+4+1] = keyer;

            _packetBuffer[12+_cBBO+4+4+4] = highByte(clip);
            _packetBuffer[12+_cBBO+4+4+5] = lowByte(clip);

            _finishCommandPacket();

        }

        /**
         * Set Downstream Keyer; Gain
         * keyer    0-3: Keyer 1-4
         * gain     0-1000: 0-100%
         */
        void ATEMstd::setDownstreamKeyerGain(uint8_t keyer, uint16_t
gain) {

            _prepareCommandPacket(PSTR("CDsG"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

                // Set Mask: 4
            _packetBuffer[12+_cBBO+4+4+0] |= 4;

            _packetBuffer[12+_cBBO+4+4+1] = keyer;

            _packetBuffer[12+_cBBO+4+4+6] = highByte(gain);
            _packetBuffer[12+_cBBO+4+4+7] = lowByte(gain);

            _finishCommandPacket();
```

```
            }

            /**
             * Set Downstream Keyer; Invert Key(??)
             * keyer    0-3: Keyer 1-4
             * invertKey    Bit 0: On/Off
             */
            void ATEMstd::setDownstreamKeyerInvertKey(uint8_t keyer, bool
 invertKey) {

                _prepareCommandPacket(PSTR("CDsG"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

                    // Set Mask: 8
                _packetBuffer[12+_cBBO+4+4+0] |= 8;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+8] = invertKey;

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Masked
             * keyer    0-3: Keyer 1-4
             * masked   Bit 0: On/Off
             */
            void ATEMstd::setDownstreamKeyerMasked(uint8_t keyer, bool ma
sked) {

                _prepareCommandPacket(PSTR("CDsM"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+2] = masked;

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Top
             * keyer    0-3: Keyer 1-4
```

```
             * top  -9000-9000: -9.00-9.00
             */
            void ATEMstd::setDownstreamKeyerTop(uint8_t keyer, int16_t to
p) {

                _prepareCommandPacket(PSTR("CDsM"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(top);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(top);

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Bottom
             * keyer    0-3: Keyer 1-4
             * bottom   -9000-9000: -9.00-9.00
             */
            void ATEMstd::setDownstreamKeyerBottom(uint8_t keyer, int16_t
 bottom) {

                _prepareCommandPacket(PSTR("CDsM"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+0] |= 4;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+6] = highByte(bottom);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(bottom);

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Left
             * keyer    0-3: Keyer 1-4
             * left     -16000-16000: -9.00-9.00
             */
```

```
        void ATEMstd::setDownstreamKeyerLeft(uint8_t keyer, int16_t l
eft) {

            _prepareCommandPacket(PSTR("CDsM"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

                // Set Mask: 8
            _packetBuffer[12+_cBBO+4+4+0] |= 8;

            _packetBuffer[12+_cBBO+4+4+1] = keyer;

            _packetBuffer[12+_cBBO+4+4+8] = highByte(left);
            _packetBuffer[12+_cBBO+4+4+9] = lowByte(left);

            _finishCommandPacket();

        }

        /**
         * Set Downstream Keyer; Right
         * keyer    0-3: Keyer 1-4
         * right    -16000-16000: -9.00-9.00
         */
        void ATEMstd::setDownstreamKeyerRight(uint8_t keyer, int16_t
right) {

            _prepareCommandPacket(PSTR("CDsM"),12,(_packetBuffer[12+_
cBBO+4+4+1]==keyer));

                // Set Mask: 16
            _packetBuffer[12+_cBBO+4+4+0] |= 16;

            _packetBuffer[12+_cBBO+4+4+1] = keyer;

            _packetBuffer[12+_cBBO+4+4+10] = highByte(right);
            _packetBuffer[12+_cBBO+4+4+11] = lowByte(right);

            _finishCommandPacket();

        }

        /**
         * Set Downstream Keyer Auto; Keyer
         * keyer    0: DSK1, 1: DSK2
         */
        void ATEMstd::performDownstreamKeyerAutoKeyer(uint8_t keyer)
{

            _prepareCommandPacket(PSTR("DDsA"),4);
```

```
    _packetBuffer[12+_cBBO+4+4+0] = keyer;

    _finishCommandPacket();

}

/**
 * Get Downstream Keyer; On Air
 * keyer    0: DSK1, 1: DSK2
 */
bool ATEMstd::getDownstreamKeyerOnAir(uint8_t keyer) {
    return atemDownstreamKeyerOnAir[keyer];
}

/**
 * Get Downstream Keyer; In Transition
 * keyer    0: DSK1, 1: DSK2
 */
bool ATEMstd::getDownstreamKeyerInTransition(uint8_t keyer) {
    return atemDownstreamKeyerInTransition[keyer];
}

/**
 * Get Downstream Keyer; Is Auto Transitioning
 * keyer    0: DSK1, 1: DSK2
 */
bool ATEMstd::getDownstreamKeyerIsAutoTransitioning(uint8_t k
eyer) {
    return atemDownstreamKeyerIsAutoTransitioning[keyer];
}

/**
 * Get Downstream Keyer; Frames Remaining
 * keyer    0: DSK1, 1: DSK2
 */
uint8_t ATEMstd::getDownstreamKeyerFramesRemaining(uint8_t ke
yer) {
    return atemDownstreamKeyerFramesRemaining[keyer];
}

/**
 * Set Downstream Keyer; On Air
 * keyer    0: DSK1, 1: DSK2
 * onAir    Bit 0: On/Off
 */
void ATEMstd::setDownstreamKeyerOnAir(uint8_t keyer, bool onA
ir) {
```

```
        _prepareCommandPacket(PSTR("CDsL"),4,(_packetBuffer[12+_c
BBO+4+4+0]==keyer));

        _packetBuffer[12+_cBBO+4+4+0] = keyer;

        _packetBuffer[12+_cBBO+4+4+1] = onAir;

        _finishCommandPacket();

    }

    /**
     * Get Fade-To-Black; Rate
     * mE    0: ME1, 1: ME2
     */
    uint8_t ATEMstd::getFadeToBlackRate(uint8_t mE) {
        return atemFadeToBlackRate[mE];
    }

    /**
     * Set Fade-To-Black; Rate
     * mE    0: ME1, 1: ME2
     * rate      1-250: Frames
     */
    void ATEMstd::setFadeToBlackRate(uint8_t mE, uint8_t rate) {

        _prepareCommandPacket(PSTR("FtbC"),4,(_packetBuffer[12+_c
BBO+4+4+1]==mE));

            // Set Mask: 1
        _packetBuffer[12+_cBBO+4+4+0] |= 1;

        _packetBuffer[12+_cBBO+4+4+1] = mE;

        _packetBuffer[12+_cBBO+4+4+2] = rate;

        _finishCommandPacket();

    }

    /**
     * Get Fade-To-Black State; Fully Black
     * mE    0: ME1, 1: ME2
     */
    bool ATEMstd::getFadeToBlackStateFullyBlack(uint8_t mE) {
        return atemFadeToBlackStateFullyBlack[mE];
    }

    /**
```

```
             * Get Fade-To-Black State; In Transition
             * mE    0: ME1, 1: ME2
             */
            bool ATEMstd::getFadeToBlackStateInTransition(uint8_t mE) {
                return atemFadeToBlackStateInTransition[mE];
            }

            /**
             * Get Fade-To-Black State; Frames Remaining
             * mE    0: ME1, 1: ME2
             */
            uint8_t ATEMstd::getFadeToBlackStateFramesRemaining(uint8_t m
E) {
                return atemFadeToBlackStateFramesRemaining[mE];
            }



                /**
                 * Set Fade-To-Black; M/E
                 * mE    0: ME1, 1: ME2
                 */
                void ATEMstd::performFadeToBlackME(uint8_t mE) {

                    _prepareCommandPacket(PSTR("FtbA"),4);

                    _packetBuffer[12+_cBBO+4+4+0] = mE;
                    _packetBuffer[12+_cBBO+4+4+1] = 0x02;

                    _finishCommandPacket();

                }

            /**
             * Set Color Generator; Hue
             * colorGenerator    0: Color Generator 1, 1: Color Generator
2

             * hue   0-3599: 0-359.9 Degrees
             */
            void ATEMstd::setColorGeneratorHue(uint8_t colorGenerator, ui
nt16_t hue) {

                _prepareCommandPacket(PSTR("CClV"),8,(_packetBuffer[12+_c
BBO+4+4+1]==colorGenerator));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = colorGenerator;
```

```
		_packetBuffer[12+_cBBO+4+4+2] = highByte(hue);
		_packetBuffer[12+_cBBO+4+4+3] = lowByte(hue);

		_finishCommandPacket();

	}

	/**
	 * Set Color Generator; Saturation
	 * colorGenerator	0: Color Generator 1, 1: Color Generator 
2
	 * saturation	0-1000: 0-100.0%
	 */
	void ATEMstd::setColorGeneratorSaturation(uint8_t colorGenera
tor, uint16_t saturation) {

		_prepareCommandPacket(PSTR("CClV"),8,(_packetBuffer[12+_c
BBO+4+4+1]==colorGenerator));

			// Set Mask: 2
		_packetBuffer[12+_cBBO+4+4+0] |= 2;

		_packetBuffer[12+_cBBO+4+4+1] = colorGenerator;

		_packetBuffer[12+_cBBO+4+4+4] = highByte(saturation);
		_packetBuffer[12+_cBBO+4+4+5] = lowByte(saturation);

		_finishCommandPacket();

	}

	/**
	 * Set Color Generator; Luma
	 * colorGenerator	0: Color Generator 1, 1: Color Generator 
2
	 * luma	0-1000: 0-100.0%
	 */
	void ATEMstd::setColorGeneratorLuma(uint8_t colorGenerator, u
int16_t luma) {

		_prepareCommandPacket(PSTR("CClV"),8,(_packetBuffer[12+_c
BBO+4+4+1]==colorGenerator));

			// Set Mask: 4
		_packetBuffer[12+_cBBO+4+4+0] |= 4;

		_packetBuffer[12+_cBBO+4+4+1] = colorGenerator;
```

```
    _packetBuffer[12+_cBBO+4+4+6] = highByte(luma);
    _packetBuffer[12+_cBBO+4+4+7] = lowByte(luma);

    _finishCommandPacket();

}

/**
 * Get Aux Source; Input
 * aUXChannel   0-5: Aux 1-6
 */
uint16_t ATEMstd::getAuxSourceInput(uint8_t aUXChannel) {
    return atemAuxSourceInput[aUXChannel];
}

/**
 * Set Aux Source; Input
 * aUXChannel   0-5: Aux 1-6
 * input     (See video source list)
 */
void ATEMstd::setAuxSourceInput(uint8_t aUXChannel, uint16_t
input) {

    _prepareCommandPacket(PSTR("CAuS"),4,(_packetBuffer[12+_c
BBO+4+4+1]==aUXChannel));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+1] = aUXChannel;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(input);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(input);

    _finishCommandPacket();

}

/**
 * Set Clip Player; Playing
 * mediaPlayer  0: Media Player 1, 1: Media Player 2
 * playing   Bit 0: On/Off
 */
void ATEMstd::setClipPlayerPlaying(uint8_t mediaPlayer, bool
playing) {

    _prepareCommandPacket(PSTR("SCPS"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mediaPlayer));
```

```
				// Set Mask: 1
			_packetBuffer[12+_cBBO+4+4+0] |= 1;

			_packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

			_packetBuffer[12+_cBBO+4+4+2] = playing;

			_finishCommandPacket();

		}

		/**
		 * Set Clip Player; Loop
		 * mediaPlayer  0: Media Player 1, 1: Media Player 2
		 * loop      Bit 0: On/Off
		 */
		void ATEMstd::setClipPlayerLoop(uint8_t mediaPlayer, bool loo
p) {

			_prepareCommandPacket(PSTR("SCPS"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mediaPlayer));

				// Set Mask: 2
			_packetBuffer[12+_cBBO+4+4+0] |= 2;

			_packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

			_packetBuffer[12+_cBBO+4+4+3] = loop;

			_finishCommandPacket();

		}

		/**
		 * Set Clip Player; At Beginning
		 * mediaPlayer  0: Media Player 1, 1: Media Player 2
		 * atBeginning  Bit 0: On/Off
		 */
		void ATEMstd::setClipPlayerAtBeginning(uint8_t mediaPlayer, b
ool atBeginning) {

			_prepareCommandPacket(PSTR("SCPS"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mediaPlayer));

				// Set Mask: 4
			_packetBuffer[12+_cBBO+4+4+0] |= 4;

			_packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;
```

```
            _packetBuffer[12+_cBBO+4+4+4] = atBeginning;

            _finishCommandPacket();

        }

        /**
         * Set Clip Player; Clip Frame
         * mediaPlayer  0: Media Player 1, 1: Media Player 2
         * clipFrame
         */
        void ATEMstd::setClipPlayerClipFrame(uint8_t mediaPlayer, uin
t16_t clipFrame) {

            _prepareCommandPacket(PSTR("SCPS"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mediaPlayer));

                // Set Mask: 8
            _packetBuffer[12+_cBBO+4+4+0] |= 8;

            _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

            _packetBuffer[12+_cBBO+4+4+6] = highByte(clipFrame);
            _packetBuffer[12+_cBBO+4+4+7] = lowByte(clipFrame);

            _finishCommandPacket();

        }

        /**
         * Get Media Player Source; Type
         * mediaPlayer  0: Media Player 1, 1: Media Player 2
         */
        uint8_t ATEMstd::getMediaPlayerSourceType(uint8_t mediaPlayer
) {
            return atemMediaPlayerSourceType[mediaPlayer];
        }

        /**
         * Get Media Player Source; Still Index
         * mediaPlayer  0: Media Player 1, 1: Media Player 2
         */
        uint8_t ATEMstd::getMediaPlayerSourceStillIndex(uint8_t media
Player) {
            return atemMediaPlayerSourceStillIndex[mediaPlayer];
        }

        /**
         * Get Media Player Source; Clip Index
```

```
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             */
            uint8_t ATEMstd::getMediaPlayerSourceClipIndex(uint8_t mediaP
layer) {
                return atemMediaPlayerSourceClipIndex[mediaPlayer];
            }

            /**
             * Set Media Player Source; Type
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             * type     1: Still, 2: Clip
             */
            void ATEMstd::setMediaPlayerSourceType(uint8_t mediaPlayer, u
int8_t type) {

                _prepareCommandPacket(PSTR("MPSS"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mediaPlayer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

                _packetBuffer[12+_cBBO+4+4+2] = type;

                _finishCommandPacket();

            }

            /**
             * Set Media Player Source; Still Index
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             * stillIndex   0-x: Still 1-x
             */
            void ATEMstd::setMediaPlayerSourceStillIndex(uint8_t mediaPla
yer, uint8_t stillIndex) {

                _prepareCommandPacket(PSTR("MPSS"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mediaPlayer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

                _packetBuffer[12+_cBBO+4+4+3] = stillIndex;

                _finishCommandPacket();
```

```
	}

	/**
	 * Set Media Player Source; Clip Index
	 * mediaPlayer  0: Media Player 1, 1: Media Player 2
	 * clipIndex    0-x: Clip 1-x
	 */
	void ATEMstd::setMediaPlayerSourceClipIndex(uint8_t mediaPlay
er, uint8_t clipIndex) {

		_prepareCommandPacket(PSTR("MPSS"),8,(_packetBuffer[12+_c
BBO+4+4+1]==mediaPlayer));

			// Set Mask: 4
		_packetBuffer[12+_cBBO+4+4+0] |= 4;

		_packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

		_packetBuffer[12+_cBBO+4+4+4] = clipIndex;

		_finishCommandPacket();

	}

	/**
	 * Get Macro Run Status; State
	 */
	uint8_t ATEMstd::getMacroRunStatusState() {
		return atemMacroRunStatusState;
	}

	/**
	 * Get Macro Run Status; Is Looping
	 */
	bool ATEMstd::getMacroRunStatusIsLooping() {
		return atemMacroRunStatusIsLooping;
	}

	/**
	 * Get Macro Run Status; Index
	 */
	uint16_t ATEMstd::getMacroRunStatusIndex() {
		return atemMacroRunStatusIndex;
	}

	/**
	 * Set Macro Action; Action
	 * index    0-99: Macro Index Number. 0xFFFF: stop
```

```
             * action   0: Run Macro, 1: Stop (w/Index 0xFFFF), 2: Stop R
ecording (w/Index 0xFFFF), 3: Insert Wait for User (w/Index 0xFFFF), 4: C
ontinue (w/Index 0xFFFF), 5: Delete Macro
             */
            void ATEMstd::setMacroAction(uint16_t index, uint8_t action)
{

                _prepareCommandPacket(PSTR("MAct"),4,(_packetBuffer[12+_c
BBO+4+4+0]==highByte(index)) && (_packetBuffer[12+_cBBO+4+4+1]==lowByte(i
ndex)));

                _packetBuffer[12+_cBBO+4+4+0] = highByte(index);
                _packetBuffer[12+_cBBO+4+4+1] = lowByte(index);

                _packetBuffer[12+_cBBO+4+4+2] = action;

                _finishCommandPacket();

            }

            /**
             * Get Macro Properties; Is Used
             * macroIndex   0-9: Macro Index Number
             */
            bool ATEMstd::getMacroPropertiesIsUsed(uint8_t macroIndex) {
                return atemMacroPropertiesIsUsed[macroIndex];
            }

            /**
             * Get Macro Properties; Name
             * macroIndex   0-9: Macro Index Number
             */
            char *  ATEMstd::getMacroPropertiesName(uint8_t macroIndex) {
                return atemMacroPropertiesName[macroIndex];
            }

            /**
             * Set Macro Add Pause; Frames
             * frames   Number of
             */
            void ATEMstd::setMacroAddPauseFrames(uint16_t frames) {

                _prepareCommandPacket(PSTR("MSlp"),4);

                _packetBuffer[12+_cBBO+4+4+2] = highByte(frames);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(frames);

                _finishCommandPacket();
```

```
}

/**
 * Get Macro Recording Status; Is Recording
 */
bool ATEMstd::getMacroRecordingStatusIsRecording() {
    return atemMacroRecordingStatusIsRecording;
}

/**
 * Get Macro Recording Status; Index
 */
uint16_t ATEMstd::getMacroRecordingStatusIndex() {
    return atemMacroRecordingStatusIndex;
}

/**
 * Get Audio Mixer Input; Mix Option
 * audioSource  (See audio source list)
 */
uint8_t ATEMstd::getAudioMixerInputMixOption(uint16_t audioSo
urce) {
    return atemAudioMixerInputMixOption[getAudioSrcIndex(audi
oSource)];
}

/**
 * Get Audio Mixer Input; Volume
 * audioSource  (See audio source list)
 */
uint16_t ATEMstd::getAudioMixerInputVolume(uint16_t audioSour
ce) {
    return atemAudioMixerInputVolume[getAudioSrcIndex(audioSo
urce)];
}

/**
 * Get Audio Mixer Input; Balance
 * audioSource  (See audio source list)
 */
int16_t ATEMstd::getAudioMixerInputBalance(uint16_t audioSour
ce) {
    return atemAudioMixerInputBalance[getAudioSrcIndex(audioS
ource)];
}

/**
 * Set Audio Mixer Input; Mix Option
 * audioSource  (See audio source list)
```

```
             * mixOption    0: Off, 1: On, 2: AFV
             */
            void ATEMstd::setAudioMixerInputMixOption(uint16_t audioSourc
e, uint8_t mixOption) {

                _prepareCommandPacket(PSTR("CAMI"),12,(_packetBuffer[12+_
cBBO+4+4+2]==highByte(audioSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lo
wByte(audioSource)));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(audioSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(audioSource);

                _packetBuffer[12+_cBBO+4+4+4] = mixOption;

                _finishCommandPacket();

            }

            /**
             * Set Audio Mixer Input; Volume
             * audioSource  (See audio source list)
             * volume   0-65381: (DB)
             */
            void ATEMstd::setAudioMixerInputVolume(uint16_t audioSource,
uint16_t volume) {

                _prepareCommandPacket(PSTR("CAMI"),12,(_packetBuffer[12+_
cBBO+4+4+2]==highByte(audioSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lo
wByte(audioSource)));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(audioSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(audioSource);

                _packetBuffer[12+_cBBO+4+4+6] = highByte(volume);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(volume);

                _finishCommandPacket();

            }

            /**
             * Set Audio Mixer Input; Balance
             * audioSource  (See audio source list)
```

```
 * balance  -10000-10000: Left/Right Extremes
 */
void ATEMstd::setAudioMixerInputBalance(uint16_t audioSource, int16_t balance) {

    _prepareCommandPacket(PSTR("CAMI"),12,(_packetBuffer[12+_cBBO+4+4+2]==highByte(audioSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lowByte(audioSource)));

        // Set Mask: 4
    _packetBuffer[12+_cBBO+4+4+0] |= 4;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(audioSource);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(audioSource);

    _packetBuffer[12+_cBBO+4+4+8] = highByte(balance);
    _packetBuffer[12+_cBBO+4+4+9] = lowByte(balance);

    _finishCommandPacket();

}

/**
 * Set Audio Mixer Master; Volume
 * volume   0-65381: (DB)
 */
void ATEMstd::setAudioMixerMasterVolume(uint16_t volume) {

    _prepareCommandPacket(PSTR("CAMM"),8);

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(volume);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(volume);

    _finishCommandPacket();

}

/**
 * Set Audio Levels; Enable
 * enable   Bit 0: On/Off
 */
void ATEMstd::setAudioLevelsEnable(bool enable) {

    _prepareCommandPacket(PSTR("SALN"),4);

    _packetBuffer[12+_cBBO+4+4+0] = enable;
```

```
    _finishCommandPacket();

}

/**
 * Get Tally By Index; Sources
 */
uint16_t ATEMstd::getTallyByIndexSources() {
    return atemTallyByIndexSources;
}

/**
 * Get Tally By Index; Tally Flags
 * sources  0-20: Number of
 */
uint8_t ATEMstd::getTallyByIndexTallyFlags(uint16_t sources)
{
    return atemTallyByIndexTallyFlags[sources];
}
```

Annex 3

ATEMbase.h (llibreria C++)

```
/*
Copyright 2012-2014 Kasper Skårhøj, SKAARHOJ K/S, kasper@skaarhoj.com

This file is part of the Blackmagic Design ATEM Client library for Arduin
o

The ATEM library is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by the
Free Software Foundation, either version 3 of the License, or (at your
option) any later version.

The ATEM library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILIT
Y
or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with the ATEM library. If not, see http://www.gnu.org/licenses/.


IMPORTANT: If you want to use this library in your own projects and/or pr
oducts,
please play a fair game and heed the license rules! See our web page for
a Q&A so
you can keep a clear conscience: http://skaarhoj.com/about/licenses/


*/




#ifndef ATEMbase_h
#define ATEMbase_h


#include "stdint.h"
#include "stdio.h"
#include "string.h"
```

```
#include "sys/select.h"
#include <netinet/in.h>

#define ATEM_headerCmd_AckRequest 0x1
// Please acknowledge reception of this package...
#define ATEM_headerCmd_HelloPacket 0x2
#define ATEM_headerCmd_Resend 0x4
// This is a resent information
#define ATEM_headerCmd_RequestNextAfter 0x8
// I'm requesting you to resend something to me.
#define ATEM_headerCmd_Ack 0x10
// This package is an acknowledge to package id (byte 4-
5) ATEM_headerCmd_AckRequest

#define ATEM_maxInitPackageCount 40
// The maximum number of initialization packages. By observation on a 2M/
E 4K can be up to (not fixed!) 32. We allocate a f more then...
#define ATEM_packetBufferLength 96
// Size of packet buffer
#define UDP_MAXSIZE 2000

#define ATEM_debug 0
// If "1" (true), more debugging information may hit the serial monitor,
in particular when _serialDebug = 0x80. Setting this to "0" is recommende
d for production environments since it saves on flash memory.

class ATEMbase
{

  protected:
    int sockfd;
    struct sockaddr_in _switcherIP;
    // IP address of the switcher
    int numbytes;
    uint16_t _localPort;
    // Default local port to send from. Preferably it's chosen randomly i
nside the class.
    char ipaddr [15];
    uint8_t _serialOutput;
    // If set, the library will print status/debug information to the Ser
ial object

    unsigned long long _initialtime;

    uint16_t _pointerbuffer;

    int sizepacketselect;

    int nfds;
```

```
    fd_set readset;

    bool _sockconnected;

    // ATEM Connection Basics
    uint16_t _localPacketIdCounter;
    // This is our counter for the command packages we might like to send
 to ATEM
    bool _initPayloadSent;
    // If true, the initial reception of the ATEM memory has passed and w
e can begin to respond during the runLoop()
    uint8_t _initPayloadSentAtPacketId;
    // The Remote Package ID at which point the initialization payload wa
s completed.
    bool _hasInitialized;
    // If true, all initial payload packets has been received during requ
ests for resent - and we are completely ready to rock!
    bool _isConnected;
    bool _waitAckSetter;
    // Set true if we have received a hello package from the switcher.
    uint16_t _sessionID;
    uint16_t _packetID;
    // Session id of session, given by ATEM switcher
    unsigned long long _lastContact;
    // Last time (millis) the switcher sent a packet to us.
    uint16_t _lastRemotePacketID;
    // The most recent Remote Packet Id from switcher
    uint8_t _missedInitializationPackages[(ATEM_maxInitPackageCount+7)/8]
;
    // Used to track which initialization packages have been missed
    uint16_t _returnPacketLength;

    // ATEM Buffer:
    uint8_t _packetBuffer[ATEM_packetBufferLength];
    uint8_t _packetIniHello[20];
    // Buffer for storing segments of the packets from ATEM and creating
answer packets.
    uint8_t _buffertotal[UDP_MAXSIZE];

    uint16_t _cmdLength;
    // Used when parsing packets
    uint16_t _cmdPointer;
    // Used when parsing packets

    bool _cBundle;
    // If set, we are building a set-command bundle.
    uint8_t _cBBO;
    // Bundle Buffer Offset; This is an offset if you want to add more co
mmands.
```

```
  uint8_t _ATEMmodel;

  bool neverConnected;
  bool waitingForIncoming;

public:
  ATEMbase();
  virtual void proces(const char *cmd);
  void begin(const char ipaddr [15]);
  void begin(const char ipaddr [15], const uint16_t localPort);
  void connecto(bool firstConnect);
  void connecto(const bool useFixedPortNumber,bool firstConnect);
  void runLoop();
  void runLoop(uint16_t delayTime);

  void readUDP(uint16_t readbytes);



  uint16_t getATEM_lastRemotePacketId();

  bool getinitPayloadSent();
  uint16_t getSessionID();

  bool isConnected();
  bool hasInitialized();

  void serialOutput(uint8_t level);
  bool hasTimedOut(unsigned long time, unsigned long timeout);

  float audioWord2Db(uint16_t input);
  uint16_t audioDb2Word(float input);

  uint8_t getVideoSrcIndex(uint16_t videoSrc);
  uint8_t getAudioSrcIndex(uint16_t audioSrc);

  uint16_t getVideoIndexSrc(uint8_t index);
  uint16_t getAudioIndexSrc(uint8_t index);

  uint8_t maxAtemSeriesVideoInputs();

  void commandBundleStart();
  void commandBundleEnd();
  void resetCommandBundle();

  uint8_t getATEMmodel();

protected:
```

```
    void _createCommandHeader(const uint8_t headerCmd, const uint16_t len
gthOfData);
    void _createCommandHeader(const uint8_t headerCmd, const uint16_t len
gthOfData, const uint16_t remotePacketID);
    void _sendPacketBuffer(uint8_t length);
    void _wipeCleanPacketBuffer();

    void _parsePacket(uint16_t packetLength);
    virtual void _parseGetCommands(const uint8_t *cmdString);
    bool _readToPacketBuffer();
    bool _readToPacketBuffer(uint8_t maxBytes);
    void _prepareCommandPacket(const char *cmdString, uint8_t cmdBytes, b
ool indexMatch=true);
    void _finishCommandPacket();
};

#endif
```

Annex 4

ATEMbase.cpp (llibreria C++)

```
/*
Copyright 2012-2014 Kasper Skårhøj, SKAARHOJ K/S, kasper@skaarhoj.com

This file is part of the Blackmagic Design ATEM Client library for Arduin
o

The ATEM library is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by the
Free Software Foundation, either version 3 of the License, or (at your
option) any later version.

The ATEM library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILIT
Y
or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with the ATEM library. If not, see http://www.gnu.org/licenses/.


IMPORTANT: If you want to use this library in your own projects and/or pr
oducts,
please play a fair game and heed the license rules! See our web page for
a Q&A so
you can keep a clear conscience: http://skaarhoj.com/about/licenses/


*/


/* Coses fetes:
El flush de dades ara nomes necessita buidar _buffertotal
Cambiat els binaris començats per B a 0B ja que es com funciona gcc
Creada una funcio per a word highbyte i lowbyte, també he modificat les f
uncions que demanaven un word per a un uint16 ja que son compatibles
Arreglat tema de strcpy, strlen i strcmp, a arduino utilitza un especial
per a no utilitzar memoria extra, no tenim aquest problema aqui
Modificat el enviament de dades per a que funcioni amb sockets
Agregat les llibreries necessaries per a que sigui tot compatible
Traduida la funcio de millis() de arduino a c
```

```
Traduida la funcio per llegir els paquets UDP i que els llegeixi a la man
era que es feia amb arduino
Agregat un select abans del read de dades per no estar enviant dades sens
e haver-ne rebut cap
Agregat un timeout al select per a que no estigui parant envios de dades,
 o si dona error
Modificat els prints de debug per a que siguin en c++

Tot el codi fet en arduino traduit a c/c++
*/



#include "ATEMbase.h"
#include "stdint.h"
#include "string.h"
#include "stdio.h"
//#include "ATEMstd.cpp"

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>
#include <math.h>
#include <iostream>
#include <sys/time.h>
#include <inttypes.h>
#include <algorithm>
using namespace std;

//ATEMstd stdV;

uint16_t word(uint8_t part1, uint8_t part2){

    uint16_t complete = part2 | part1 << 8;

    return complete;
}

void ATEMbase::proces(const char *cmd){
    cout <<" PROBLEMA PRCES" ;
```

```
};
uint8_t lowByte(uint16_t palabra){
    uint8_t low = palabra & 0xFF;
    return low;
}

uint8_t highByte(uint16_t palabra){
    uint8_t high = (palabra >> 8) & 0xFF;
    return high;
}

unsigned long long current_timestamp() {
    struct timeval te;
    gettimeofday(&te, NULL);
    unsigned long long milliseconds = te.tv_sec*1000LL + te.tv_usec/1000;
    return milliseconds;
}

unsigned long long millis(unsigned long long _initialtime){
    return current_timestamp() - _initialtime;
}


void ATEMbase::readUDP(uint16_t readbytes){
    //Buidar packetbuffer
    cout <<" readbytes : " <<readbytes <<" sizePacketselect: "<< sizepack
etselect << " ";
    std::fill_n(_packetBuffer, ATEM_packetBufferLength, 0);
    //funcio d'omplir packetbuffer amb bytes buffertotal
    if(_pointerbuffer + readbytes <= sizepacketselect){
        for(uint16_t i = 0; i < readbytes; i++){
            _packetBuffer[i] = _buffertotal[i+_pointerbuffer];
        }
        _pointerbuffer += readbytes;
        //Esta be
    } else{
        printf("%s \n", "Ha habido un error en la lectura en readUDP");
        //error
    }

}


ATEMbase::ATEMbase(){}

/**
 * Setting up IP address for the switcher (and local port to send packets
 from)
```

```
 * Using local port here is deprecated. Rather let the library pick a ran
dom one
 */
void ATEMbase::begin(const char ip [15]){
    begin(ip, /*(50100 + (std::rand() % (65300 - 50100 + 1)))*/ 9910);
}
void ATEMbase::begin(const char ip [15], const uint16_t localPort){

    _initialtime = current_timestamp();

    neverConnected = true;
    waitingForIncoming = false;

    ipaddr[15] = ip[15];

    _localPort = localPort;
    //cout << "local port: "<<localPort;
    // Set default local port

    // Set up Udp communication object:

    _switcherIP.sin_family = AF_INET;
    /* host byte order */
    _switcherIP.sin_port = htons(_localPort);
    /* short, network byte order */
    inet_aton(ip,&_switcherIP.sin_addr);

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
            perror("Error Generació de socket");
    }

    nfds = sockfd +1;

    fcntl(sockfd, F_SETFL, O_NONBLOCK);

    _lastContact = 0;
    _serialOutput = 0;

    resetCommandBundle();
}

/**
 * Initiating connection handshake to the ATEM switcher
 */
void ATEMbase::connecto(bool firstTime) {
    connecto(false, firstTime);
}

/**
```

```
 * Initiating connection handshake to the ATEM switcher
 * If useFixedPortNumber is true, the same port number will be used on su
bsequent connects, otherwise - and recommended - a new, random port numbe
r is used.
 */
void ATEMbase::connecto(const bool useFixedPortNumber, bool firstTime) {
    _localPacketIdCounter = 0;
    // Init localPacketIDCounter to 0;
    _initPayloadSent = false;
    // Will be true after initial payload of data is delivered (regular 1
2-byte ping packages are transmitted.)
    _hasInitialized = false;
    // Will be true after initial payload of data is resent and received
well
    _isConnected = false;
    _sockconnected = false;
    _waitAckSetter= false;
    // Will be true after the initial hello-package handshakes.
    _sessionID = 0x53AB;
    // Temporary session ID - a new will be given back from ATEM.
    _lastContact = millis(_initialtime);
    // Setting this, because even though we haven't had contact, it const
itutes an attempt that should be responded to at least
    memset(_missedInitializationPackages, 0xFF, (ATEM_maxInitPackageCount
+7)/8);
    _initPayloadSentAtPacketId = ATEM_maxInitPackageCount;
    // The max value it can be
    uint16_t portNumber = useFixedPortNumber ? _localPort : (50100 + (std
::rand() % (65300 - 50100 + 1)));


    if((connect(sockfd, (struct sockaddr *)&_switcherIP, sizeof(_switcher
IP))) < 0){
            perror("Error in connect");

        }

    _sockconnected = true;

    // Send connectString to ATEM:
    if (_serialOutput)  {
        printf("%s \n", "Sending connect packet to ATEM switcher on IP ")
;
        printf("%s\n", ipaddr);
        printf("%s \n", " from port ");
        printf("%" PRIu16 "\n",_localPort);
    }

    _wipeCleanPacketBuffer();
```

```
    _createCommandHeader(ATEM_headerCmd_HelloPacket, 12+8);
    _packetBuffer[12] = 0x01;
    // This seems to be what the client should send upon first request.
    _packetBuffer[9] = 0x3a;
    // This seems to be what the client should send upon first request.
    //printf(" 1er packBuf i");
    if(firstTime){
    _sendPacketBuffer(20);}
    //printf(" 1er packBuf f");
}

/**
 * Keeps connection to the switcher alive
 * Therefore: Call this in the Arduino loop() function and make sure it g
ets call at least 2 times a second
 * Other recommendations might come up in the future.
 */
void ATEMbase::runLoop() {
    runLoop(0);
}
void ATEMbase::runLoop(uint16_t delayTime) {
    if (neverConnected) {
        neverConnected = false;
        connecto(true);
        printf("Connecting first time...");
    }

    unsigned long long enterTime = millis(_initialtime);
    struct timeval timeout;

        while(true) {
            // Iterate until UDP buffer is empty



                FD_ZERO(&readset);

                FD_SET(sockfd, &readset);

                int n;

                timeout.tv_sec = floor(delayTime/1000);

                //timeout.tv_usec = (delayTime/1000 - floor(delayTime/100
0))*1000000;
                if((n = select(nfds, &readset, NULL, NULL, &timeout)) ==
-1){
                    perror("select");
                }
```

```
                cout <<" / n= " << n<< " " ;
            if (n != 0) {
                /*if (n == 0){
                    printf("Timeout select");
                    connecto(false);
                    break;
                }*/

                if(FD_ISSET(sockfd, &readset)){
                    //Llest per ser llegit el socket
                    sizepacketselect = recv(sockfd, _buffertotal, UDP_MAX
SIZE, 0);
                }

                _pointerbuffer = 0;

                //Agafar 12 primers del buffertotal
                //cout << " "<< "read i ";
                readUDP(12);
                 _sessionID = word(_packetBuffer[2], _packetBuffer[3]);
                 uint8_t headerBitmask = _packetBuffer[0]>>3;
                 _packetID =word(_packetBuffer[4], _packetBuffer[5]);
                cout<<" header bimask: "<<headerBitmask;
                 _lastRemotePacketID = word(_packetBuffer[10],_packetBuff
er[11]);

                 if (_lastRemotePacketID < ATEM_maxInitPackageCount)    {
                    _missedInitializationPackages[_lastRemotePacketID>>3]
 &= ~(0B1<<(_lastRemotePacketID&0x07));
                 }

                 uint16_t packetLength = word(_packetBuffer[0] & 0x07, _p
acketBuffer[1]);
                 cout <<" PacketLength= "<<packetLength << " ";

                if (sizepacketselect==packetLength) {
                    //cout << " Size=plength ";
                    // Just to make sure these are equal, they should be!
                    _lastContact = millis(_initialtime);
                    waitingForIncoming = false;

                    if (headerBitmask & ATEM_headerCmd_HelloPacket) {
                        // Respond to "Hello" packages:
                        cout <<" Hello Pack ";
                        _isConnected = true;

                        // _packetBuffer[12]    The ATEM will return a "2
" in this return package of same length. If the ATEM returns "3" it means
 "fully booked" (no more clients can connect) and a "4" seems to be a kin
```

```
d of reconnect (seen when you drop the connection and the ATEM desperatel
y tries to figure out what happened...)
                        // _packetBuffer[15]    This number seems to incr
ement with about 3 each time a new client tries to connect to ATEM. It ma
y be used to judge how many client connections has been made during the u
p-time of the switcher?

                        _wipeCleanPacketBuffer();
                        _createCommandHeader(ATEM_headerCmd_Ack, 12);
                        _packetBuffer[9] = 0x03;
                        // This seems to be what the client should send u
pon first request.
                        _sendPacketBuffer(12);
                    }

                    if (!(headerBitmask & ATEM_headerCmd_HelloPacket) &&
packetLength>12 && _packetID==_localPacketIdCounter && _waitAckSetter) {
                        cout<<endl<<"..........."<<"Setter rebut correcta
ment"<<"..........."<<endl;
                        _waitAckSetter=false;
                    }
                    else if ((packetLength<=12 || _packetID!=_localPacket
IdCounter) && _waitAckSetter)  {
                        cout<<endl<<"..........."<<"Error al enviar el se
tter"<<"..........."<<endl;
                        _waitAckSetter=false;
                    }
                    // If a packet is 12 bytes long it indicates that all
 the initial information
                    // has been delivered from the ATEM and we can begin
to answer back on every request
                    // Currently we don't know any other way to decide if
 an answer should be sent back...
                    // The QT lib uses the "InCm" command to indicate thi
s, but in the latest version of the firmware (2.14)
                    // all the camera control information comes AFTER thi
s command, so it's not a clear ending token anymore.
                    // However, I'm not sure if I checked the _lastRemote
PacketID of the packages with the additional camera control info - if it
was a resend,
                    // "InCm" may still indicate the number of the last i
nit-package and that's all I need to request the missing ones....

                    // BTW: It has been observed on an old 10Mbit hub tha
t packages could arrive in a different order than sent and this may
                    // mess things up a bit on the initialization. So it'
s recommended to has as direct routes as possible.
                    if(!_initPayloadSent && sizepacketselect == 12 && _la
stRemotePacketID>1) {
```

```
                        cout << " End Payload";
                        _initPayloadSent = true;

                        _initPayloadSentAtPacketId = _lastRemotePacketID;
                        #if ATEM_debug
                        if (_serialOutput & 0x80) {
                            printf("_initPayloadSent=TRUE @rpID ");
                            printf("%" PRIu8 "\n",_initPayloadSentAtPacke
tId);
                            printf("Session ID: ");
                            printf("%" PRIu16 "\n",_sessionID);
                        }
                        #endif
                    }

                    if (_initPayloadSent && (headerBitmask & ATEM_headerC
md_AckRequest) && (_hasInitialized || !(headerBitmask & ATEM_headerCmd_Re
send))) {
                        // Respond to request for acknowledge   (and to r
esends also, whatever...
                        cout <<" ACK Request ";
                        _wipeCleanPacketBuffer();
                        _createCommandHeader(ATEM_headerCmd_Ack, 12, _las
tRemotePacketID);
                        _sendPacketBuffer(12);


                        #if ATEM_debug
                        if (_serialOutput & 0x80) {
                            printf("rpID: ");
                            printf("%" PRIu16 "\n",_lastRemotePacketID);
                            printf(", Head: 0x");
                            printf("%" PRIu8 "\n",headerBitmask);
                            printf(", Len: ");
                            printf("%" PRIu16 "\n",packetLength);
                            printf(" bytes");

                            printf(" - ACK!");
                        } else
                        #endif
                        if (_serialOutput>1)    {
                            printf("rpID: ");
                            printf("%" PRIu16 "\n",_lastRemotePacketID);
                            printf(" - ACK!");
                        }
                    } else if(_initPayloadSent && (headerBitmask & ATEM_h
eaderCmd_RequestNextAfter) && _hasInitialized) {
                        // ATEM is requesting a previously sent package w
hich must have dropped out of the order. We return an empty one so the AT
```

```
EM doesnt' crash (which some models will, if it doesn't get an answer bef
ore another 63 commands gets sent from the controller.)
                        cout <<"Resend Info Request ";
                        uint8_t b1 = _packetBuffer[6];
                        uint8_t b2 = _packetBuffer[7];
                        _wipeCleanPacketBuffer();
                        _createCommandHeader(ATEM_headerCmd_Ack, 12, 0);
                        _packetBuffer[0] = ATEM_headerCmd_AckRequest << 3
;
                        // Overruling this. A small trick because createC
ommandHeader shouldn't increment local package ID counter
                        _packetBuffer[10] = b1;
                        _packetBuffer[11] = b2;
                        _sendPacketBuffer(12);

                        if (_serialOutput>1)    {
                            printf("ATEM asking to resend ");
                            printf("%" PRIu8 "\n",(b1<<8)|b2);
                        }
                    } else {
                        #if ATEM_debug
                        if (_serialOutput & 0x80) {
                            printf("rpID: ");
                            printf("%" PRIu16 "\n",_lastRemotePacketID);
                            printf(", Head: 0x");
                            printf("%" PRIu8 "\n",headerBitmask);
                            printf(", Len: ");
                            printf("%" PRIu16 "\n",packetLength);
                            printf(" bytes");
                        } else
                        #endif
                        if (_serialOutput>1)    {
                            printf("rpID: ");
                            printf("%" PRIu16 "\n",_lastRemotePacketID);
                        }
                    }

                    if (!(headerBitmask & ATEM_headerCmd_HelloPacket) &&
packetLength>12)   {
                        _parsePacket(packetLength);
                    }
                } /*else {
                    #if ATEM_debug
                    if (_serialOutput & 0x80)   {
                        printf("ERROR: Packet size mismatch: ");
                        printf("%d\n",sizepacketselect );
                        printf(" != ");
                        printf("%" PRIu16 "\n",packetLength);
                    }
```

```
                    #endif
                    // Flushing:
                    std::fill_n(_buffertotal, UDP_MAXSIZE, 0);
                    std::fill_n(_packetBuffer, ATEM_packetBufferLength, 0
);

                }*/
            } else break;
        }

        // After initialization, we check which packages were missed and
ask for them:
        if (!_hasInitialized && _initPayloadSent && !waitingForIncoming)
   {
            cout<<" Ask Pack ";
            for(uint8_t i=1; i<_initPayloadSentAtPacketId; i++) {
                if(i <= ATEM_maxInitPackageCount) {
                    if (_missedInitializationPackages[i>>3] & (0B1<<(i &
0x7))) {

                        #if ATEM_debug
                        if (_serialOutput & 0x80)   {
                            printf("Asking for package ");
                            printf("%d\n", i);

                        }
                        #endif
                        _wipeCleanPacketBuffer();
                        _createCommandHeader(ATEM_headerCmd_RequestNextAf
ter, 12);

                        _packetBuffer[6] = highByte(i-1);
                        // Resend Packet ID, MSB
                        _packetBuffer[7] = lowByte(i-1);
                        // Resend Packet ID, LSB
                        _packetBuffer[8] = 0x01;

                        _sendPacketBuffer(12);
                        waitingForIncoming = true;
                        break;
                    }
                } else {
                    break;
                }
            }
            if (!waitingForIncoming)    {
                _hasInitialized = true;
                if (_serialOutput) {
                    printf("ATEM _hasInitialized = TRUE");
                }
```

```
            }
        }


    // If connection is gone anyway, try to reconnect:
    /* //no cal ja que ho fem al select
    if (hasTimedOut(_lastContact, 5000))    {
      if (_serialOutput) Serial.println(F("Connection to ATEM Switcher ha
s timed out - reconnecting!"));
      connecto();
    }
    */
}

/**
 * Returns last Remote Packet ID
 */
uint16_t ATEMbase::getATEM_lastRemotePacketId() {
    return _lastRemotePacketID;
}
/**
 * Get if the initial payload has been send
 */
bool ATEMbase::getinitPayloadSent(){
    return _initPayloadSent;
}
/**
 * Get ATEM session ID
 */
uint16_t ATEMbase::getSessionID() {
    return _sessionID;
}

/**
 * If true, we had a response from the switcher when trying to send a hel
lo packet.
 */
bool ATEMbase::isConnected()    {
    return _isConnected;
}

/**
 * If true, the initial handshake and "stressful" information exchange ha
s occured and now the switcher connection should be ready for operation.
 */
bool ATEMbase::hasInitialized() {
    return _hasInitialized;
}
```

```
/**************
 *
 * Buffer work
 *
 **************/

void ATEMbase::_createCommandHeader(const uint8_t headerCmd, const uint16
_t lengthOfData)   {
    _createCommandHeader(headerCmd, lengthOfData, 0);
}
void ATEMbase::_createCommandHeader(const uint8_t headerCmd, const uint16
_t lengthOfData, const uint16_t remotePacketID)    {

    _packetBuffer[0] = (headerCmd << 3) | (highByte(lengthOfData) & 0x07)
;
    // Command bits + length MSB
    _packetBuffer[1] = lowByte(lengthOfData);
    // length LSB

    _packetBuffer[2] = highByte(_sessionID);
    // Session ID
    _packetBuffer[3] = lowByte(_sessionID);
    // Session ID

    _packetBuffer[4] = highByte(remotePacketID);
    // Remote Packet ID, MSB
    _packetBuffer[5] = lowByte(remotePacketID);
    // Remote Packet ID, LSB

    if(!(headerCmd & (ATEM_headerCmd_HelloPacket | ATEM_headerCmd_Ack | A
TEM_headerCmd_RequestNextAfter))) {
        _localPacketIdCounter++;

//      if ((_localPacketIdCounter & 0xF) == 0xF) _localPacketIdCounter++
;
// Uncommenting this line will jump the local package ID counter every 15
 command - thereby introducing a stress test of the robustness of the "re
sent package" function from the ATEM switcher.

        _packetBuffer[10] = highByte(_localPacketIdCounter);
        // Local Packet ID, MSB
        _packetBuffer[11] = lowByte(_localPacketIdCounter);
        // Local Packet ID, LSB
```

```
    }
}
void ATEMbase::_sendPacketBuffer(uint8_t length)    {

    if ((numbytes=send(sockfd, _packetBuffer, length, 0)) < 0) {
            perror("sendpacketbuffer error");
        }
}

/**
 * Sets all zeros in packet buffer:
 */
void ATEMbase::_wipeCleanPacketBuffer() {
    memset(_packetBuffer, 0, ATEM_packetBufferLength);
}

/**
 * Reads from UDP channel to buffer. Will fill the buffer to the max or t
o the size of the current segment being parsed
 * Returns false if there are no more bytes, otherwise true
 */
bool ATEMbase::_readToPacketBuffer() {
    return _readToPacketBuffer(ATEM_packetBufferLength);
}
bool ATEMbase::_readToPacketBuffer(uint8_t maxBytes) {
    maxBytes = maxBytes<=ATEM_packetBufferLength ? maxBytes : ATEM_packet
BufferLength;
    int remainingBytes = _cmdLength-8-_cmdPointer;

    if (remainingBytes>0)   {
        if (remainingBytes <= maxBytes) {
            readUDP(remainingBytes);
            _cmdPointer+= remainingBytes;
            return false;
            // Returns false if finished.
        } else {
            readUDP(maxBytes);
            _cmdPointer+= maxBytes;
            return true;
            // Returns true if there are still bytes to be read.
        }
    } else {
        return false;
    }
}

/**
 * If a package longer than a normal acknowledgement is received from the
 ATEM Switcher we must read through the contents.
```

```
 * Usually such a package contains updated state information about the mi
xer
 * Selected information is extracted in this function and transferred to
internal variables in this library.
 */
void ATEMbase::_parsePacket(uint16_t packetLength)  {

        // If packet is more than an ACK packet (= if its longer than 12
bytes header), lets parse it:
      uint16_t indexPointer = 12;
      // 12 bytes has already been read from the packet...
      while (indexPointer < packetLength)  {

        // Read the length of segment (first word):
        readUDP(8);
        _cmdLength = word(_packetBuffer[0], _packetBuffer[1]);
        _cmdPointer = 0;

            // Get the "command string", basically this is the 4 char var
iable name in the ATEM memory holding the various state values of the sys
tem:
        uint8_t cmdStr[] = {
          _packetBuffer[4], _packetBuffer[5], _packetBuffer[6], _packetBu
ffer[7], '\0'};

            // If length of segment larger than 8 (should always be...!)
        if (_cmdLength>8)  {
            char* cmdChar = reinterpret_cast<char*>(cmdStr);
            cout << cmdChar;
            proces(cmdChar);
            //stdV.ParsePack(cmdChar);

            while (_readToPacketBuffer())   {}
            // Empty, if not done yet.
            indexPointer+=_cmdLength;
        } else {
            indexPointer = 2000;
            #if ATEM_debug
                if (_serialOutput & 0x80) printf("Bad CMD length, flushin
g...");
            #endif

            // Flushing the buffer:
            std::fill_n(_buffertotal, UDP_MAXSIZE, 0);
            std::fill_n(_packetBuffer, ATEM_packetBufferLength, 0);
        }
      }
}
```

```
/**
 * This method should be overloaded in subclasses in order to handle spec
ific get-commands
 */
void ATEMbase::_parseGetCommands(const uint8_t *cmdString)  {
//  uint8_t mE, keyer, mediaPlayer, aUXChannel, windowIndex, multiViewer,
 memory, colorGenerator, box;
//  uint16_t audioSource, videoSource;
//  long temp;
    cout <<" PARSE : "<< cmdString << endl;
    uint8_t numberOfReads=1;
    while(_readToPacketBuffer())    {
        numberOfReads++;
    }
    #if ATEM_debug
    if (_serialOutput & 0x80) {
        printf("%s\n", cmdString)
        printf(", len: ");
        printf("%" PRIu16 "\n", _cmdLength);
        printf(", rds: ");
        printf("%" PRIu8 "\n", numberOfReads);
    }
    #endif
}

void ATEMbase::_prepareCommandPacket(const char *cmdString, uint8_t cmdBy
tes, bool indexMatch)  {


        if (_cBundle)   {
            if (_returnPacketLength>0 && (!indexMatch || strncmp((char *)
(_packetBuffer+12+_cBBO+4), cmdString, 4))){
                _cBBO = _returnPacketLength-12;
            }
        } else {
            _wipeCleanPacketBuffer();
            // For command bundles, this is already done...
        }
    cout <<endl<<"cBBO: "<<_cBBO<< " cmdBytes:"<< cmdBytes<<endl;
      _returnPacketLength = 12+_cBBO+(4+4+cmdBytes);

      // Because we increased length of command, we need to check for buf
fer overflow:
      if (_returnPacketLength > ATEM_packetBufferLength)    {
          printf("FATAL ERROR: Packet Buffer Overflow in the ATEM Library
!");
          while(true){} // STOP!
      }
```

```
        // Copy Command String:
        if (strlen(cmdString)==4)   {
            //strncpy((char *)(_packetBuffer+12+_cBBO+4), cmdString, 4);
            for(int i=0;i<4;i++){
                _packetBuffer[12 + _cBBO +4+i]=cmdString[i];
            }
        }
        #if ATEM_debug
        else printf("Command Length > 4 ERROR");
        #endif

      // Command length:
      _packetBuffer[12+_cBBO] = 0;
      _packetBuffer[12+1+_cBBO] = 4+4+cmdBytes; // LSB

}

void ATEMbase::_finishCommandPacket()   {

    if (!_cBundle)  {

      _createCommandHeader(ATEM_headerCmd_AckRequest, _returnPacketLength
);
      cout<<endl<<"Packet_F:";
        for(int i=0; i<sizeof(_packetBuffer);i++){
            cout<<hex<<int(_packetBuffer[i]);
        }
        cout<<endl<<"P_length:"<<int(_returnPacketLength);
      _sendPacketBuffer(_returnPacketLength);
      _waitAckSetter=true;
      _returnPacketLength = 0;

    } else {
          // Debugging info:
    /*    for(uint8_t a=0; a<_returnPacketLength; a++)  {
          if (_packetBuffer[a]<16)  Serial.print("0");
              Serial.print(_packetBuffer[a], HEX);
              Serial.print(F("-"));
          }
          Serial.println();
        */
    }
}


/*************
 *
 * Utilities from SkaarhojTools class:
```

```
 *
 *************/

/**
 * Setter method: If _serialOutput is set, the library may use Serial.pri
nt() to give away information about its operation - mostly for debugging.
 * 0= no output
 * 1= normal output (info)
 * 2= verbose
 * &0x80 (bit 7 set): verbose initial connection information
 */
void ATEMbase::serialOutput(uint8_t level) {
    _serialOutput = level;
}

/**
 * Timeout check
 */
bool ATEMbase::hasTimedOut(unsigned long time, unsigned long timeout)  {
  if ((unsigned long)(time + timeout) <= (unsigned long long)millis(_init
ialtime))  {
      // This should "wrap around" if time+timout is larger than the size
 of unsigned-longs, right?
    return true;
  }
  else {
    return false;
  }
}




uint8_t ATEMbase::getATEMmodel()    {
    return _ATEMmodel;
}




float ATEMbase::audioWord2Db(uint16_t input)  {
    // -48 to +6 output
  // Formular: log10(input/128)*20-48;
  if (input<=32)  return -60;
  //return (log10(input)-2.1072099696)*20-48;
```

```
  // Better way?
  //return log10(input >> 5) * 20.0 - 60.0;

  return log10f((float)input/(1<<11) / 16.0) * 20.0;
}
uint16_t ATEMbase::audioDb2Word(float input)  {
    // -48 to +6 input
  //return (float)pow(10,(input+48)/20)*128;
    //return (uint16_t)pow(10, (input + 60.0) / 20.0) << 5;

    return pow(10, input/20.0) * 16.0 * (1<<11);
}
```

```
uint8_t ATEMbase::getVideoSrcIndex(uint16_t videoSrc)   {
    switch(videoSrc){
        case 0:  // Black
            return 0;
        case 1:  // Input 1
            return 1;
        case 2:  // Input 2
            return 2;
        case 3:  // Input 3
            return 3;
        case 4:  // Input 4
            return 4;
        case 5:  // Input 5
            return 5;
        case 6:  // Input 6
            return 6;
        case 7:  // Input 7
            return 7;
        case 8:  // Input 8
            return 8;
        case 9:  // Input 9
            return 9;
        case 10:  // Input 10
            return 10;
        case 11:  // Input 11
            return 11;
        case 12:  // Input 12
            return 12;
```

```
case 13:  // Input 13
    return 13;
case 14:  // Input 14
    return 14;
case 15:  // Input 15
    return 15;
case 16:  // Input 16
    return 16;
case 17:  // Input 17
    return 17;
case 18:  // Input 18
    return 18;
case 19:  // Input 19
    return 19;
case 20:  // Input 20
    return 20;
case 1000:  // Color Bars
    return 21;
case 2001:  // Color 1
    return 22;
case 2002:  // Color 2
    return 23;
case 3010:  // Media Player 1
    return 24;
case 3011:  // Media Player 1 Key
    return 25;
case 3020:  // Media Player 2
    return 26;
case 3021:  // Media Player 2 Key
    return 27;
case 4010:  // Key 1 Mask
    return 28;
case 4020:  // Key 2 Mask
    return 29;
case 4030:  // Key 3 Mask
    return 30;
case 4040:  // Key 4 Mask
    return 31;
case 5010:  // DSK 1 Mask
    return 32;
case 5020:  // DSK 2 Mask
    return 33;
case 6000:  // Super Source
    return 34;
case 7001:  // Clean Feed 1
    return 35;
case 7002:  // Clean Feed 2
    return 36;
case 8001:  // Auxilary 1
```

```
            return 37;
        case 8002:  // Auxilary 2
            return 38;
        case 8003:  // Auxilary 3
            return 39;
        case 8004:  // Auxilary 4
            return 40;
        case 8005:  // Auxilary 5
            return 41;
        case 8006:  // Auxilary 6
            return 42;
        case 10010:  // ME 1 Prog
            return 43;
        case 10011:  // ME 1 Prev
            return 44;
        case 10020:  // ME 2 Prog
            return 45;
        case 10021:  // ME 2 Prev
            return 46;
        default:
            return 0;
    }
}

uint8_t ATEMbase::getAudioSrcIndex(uint16_t audioSrc)   {
    switch(audioSrc){
        case 1:  // Input 1
            return 0;
        case 2:  // Input 2
            return 1;
        case 3:  // Input 3
            return 2;
        case 4:  // Input 4
            return 3;
        case 5:  // Input 5
            return 4;
        case 6:  // Input 6
            return 5;
        case 7:  // Input 7
            return 6;
        case 8:  // Input 8
            return 7;
        case 9:  // Input 9
            return 8;
        case 10:  // Input 10
            return 9;
        case 11:  // Input 11
            return 10;
        case 12:  // Input 12
```

```
            return 11;
        case 13:  // Input 13
            return 12;
        case 14:  // Input 14
            return 13;
        case 15:  // Input 15
            return 14;
        case 16:  // Input 16
            return 15;
        case 17:  // Input 17
            return 16;
        case 18:  // Input 18
            return 17;
        case 19:  // Input 19
            return 18;
        case 20:  // Input 20
            return 19;
        case 1001:  // XLR
            return 20;
        case 1101:  // AES/EBU
            return 21;
        case 1201:  // RCA
            return 22;
        case 2001:  // MP1
            return 23;
        case 2002:  // MP2
            return 24;
        default:
            return 0;
    }
}

/*
 * Translating a index to a video source
 */
uint16_t ATEMbase::getVideoIndexSrc(uint8_t index)  {
  switch (index) {
    case 0:  // Black
      return 0;
    case 1:  // Input 1
      return 1;
    case 2:  // Input 2
      return 2;
    case 3:  // Input 3
      return 3;
    case 4:  // Input 4
      return 4;
    case 5:  // Input 5
      return 5;
```

```
case 6:  // Input 6
  return 6;
case 7:  // Input 7
  return 7;
case 8:  // Input 8
  return 8;
case 9:  // Input 9
  return 9;
case 10:  // Input 10
  return 10;
case 11:  // Input 11
  return 11;
case 12:  // Input 12
  return 12;
case 13:  // Input 13
  return 13;
case 14:  // Input 14
  return 14;
case 15:  // Input 15
  return 15;
case 16:  // Input 16
  return 16;
case 17:  // Input 17
  return 17;
case 18:  // Input 18
  return 18;
case 19:  // Input 19
  return 19;
case 20:  // Input 20
  return 20;
case 21:  // Color Bars
  return 1000;
case 22:  // Color 1
  return 2001;
case 23:  // Color 2
  return 2002;
case 24:  // Media Player 1
  return 3010;
case 25:  // Media Player 1 Key
  return 3011;
case 26:  // Media Player 2
  return 3020;
case 27:  // Media Player 2 Key
  return 3021;
case 28:  // Key 1 Mask
  return 4010;
case 29:  // Key 2 Mask
  return 4020;
case 30:  // Key 3 Mask
```

```
      return 4030;
    case 31:  // Key 4 Mask
      return 4040;
    case 32:  // DSK 1 Mask
      return 5010;
    case 33:  // DSK 2 Mask
      return 5020;
    case 34:  // Super Source
      return 6000;
    case 35:  // Clean Feed 1
      return 7001;
    case 36:  // Clean Feed 2
      return 7002;
    case 37:  // Auxilary 1
      return 8001;
    case 38:  // Auxilary 2
      return 8002;
    case 39:  // Auxilary 3
      return 8003;
    case 40:  // Auxilary 4
      return 8004;
    case 41:  // Auxilary 5
      return 8005;
    case 42:  // Auxilary 6
      return 8006;
    case 43:  // ME 1 Prog
      return 10010;
    case 44:  // ME 1 Prev
      return 10011;
    case 45:  // ME 2 Prog
      return 10020;
    case 46:  // ME 2 Prev
      return 10021;
    default:
      return 0;
  }
}

/*
 * Translating a index to a audio source
 */
uint16_t ATEMbase::getAudioIndexSrc(uint8_t index)  {
  switch (index) {
    case 0:  // Input 1
      return 1;
    case 1:  // Input 2
      return 2;
    case 2:  // Input 3
      return 3;
```

```
      case 3:  // Input 4
        return 4;
      case 4:  // Input 5
        return 5;
      case 5:  // Input 6
        return 6;
      case 6:  // Input 7
        return 7;
      case 7:  // Input 8
        return 8;
      case 8:  // Input 9
        return 9;
      case 9:  // Input 10
        return 10;
      case 10:  // Input 11
        return 11;
      case 11:  // Input 12
        return 12;
      case 12:  // Input 13
        return 13;
      case 13:  // Input 14
        return 14;
      case 14:  // Input 15
        return 15;
      case 15:  // Input 16
        return 16;
      case 16:  // Input 17
        return 17;
      case 17:  // Input 18
        return 18;
      case 18:  // Input 19
        return 19;
      case 19:  // Input 20
        return 20;
      case 20:  // XLR
        return 1001;
      case 21:  // AES/EBU
        return 1101;
      case 22:  // RCA
        return 1201;
      case 23:  // MP1
        return 2001;
      case 24:  // MP2
        return 2002;
      default:
        return 0;
    }
  }
```

```
uint8_t ATEMbase::maxAtemSeriesVideoInputs()	{
	return 47;
	// For the largest ATEM switcher, this is the number of video inputs.
 The max "index" number from the list above
}


void ATEMbase::commandBundleStart() {
	resetCommandBundle();
	_wipeCleanPacketBuffer();
	_cBundle = true;
}
void ATEMbase::commandBundleEnd()	{
	if (_cBundle && _returnPacketLength > 0)	{
		
	_createCommandHeader(ATEM_headerCmd_AckRequest, _returnPacketLength
);
		_sendPacketBuffer(_returnPacketLength);
		_returnPacketLength = 0;
	}
	resetCommandBundle();
}
void ATEMbase::resetCommandBundle() {
	_cBundle = false;
	_cBBO = 0;
}
```

Annex 5

ATEMext.cpp (llibreria C++)

```
/*
Copyright 2012-2014 Kasper Skårhøj, SKAARHOJ K/S, kasper@skaarhoj.com

This file is part of the Blackmagic Design ATEM Client library for Arduin
o

The ATEM library is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by the
Free Software Foundation, either version 3 of the License, or (at your
option) any later version.

The ATEM library is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILIT
Y
or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with the ATEM library. If not, see http://www.gnu.org/licenses/.


IMPORTANT: If you want to use this library in your own projects and/or pr
oducts,
please play a fair game and heed the license rules! See our web page for
a Q&A so
you can keep a clear conscience: http://skaarhoj.com/about/licenses/


*/



#include "ATEMext.h"
#include "ATEMbaseV1.cpp"
using namespace std;

/**
 * Constructor (using arguments is deprecated! Use begin() instead)
 */
ATEMext::ATEMext(){}
void ATEMext::proces(const char *cmd){
            cout << "CMD: " << cmd;
```

```
            _parseGetCommands(cmd);
    }

uint8_t ATEMext::getTallyFlags(uint16_t videoSource)  {
  for (uint8_t a = 0; a < getTallyBySourceSources(); a++)  {
    if (getTallyBySourceVideoSource(a) == videoSource)  {
        return getTallyBySourceTallyFlags(a);
    }
  }
  return 0;
}
uint8_t ATEMext::getAudioTallyFlags(uint16_t audioSource)  {
  for (uint8_t a = 0; a < getAudioMixerTallySources(); a++)  {
    if (getAudioMixerTallyAudioSource(a) == audioSource)  {
        return getAudioMixerTallyIsMixedIn(a);
    }
  }
  return 0;
}

void ATEMext::setCameraControlVideomode(uint8_t input, uint8_t fps, uint8
_t resolution, uint8_t interlaced) {
        _prepareCommandPacket(("CCmd"), 24);

        _packetBuffer[12+_cBBO+4+4+0] = input;

        _packetBuffer[12+_cBBO+4+4+1] = 1;
        _packetBuffer[12+_cBBO+4+4+2] = 0;

        _packetBuffer[12+_cBBO+4+4+4] = 0x01; // Data type: int8

        _packetBuffer[12+_cBBO+4+4+7] = 0x05; // 5 Byte array

        //_packetBuffer[12+_cBBO+4+4+9] = 0x05; // 5 byte array

        _packetBuffer[12+_cBBO+4+4+16] = fps;
        _packetBuffer[12+_cBBO+4+4+17] = 0x00; // Regular M-rate
        _packetBuffer[12+_cBBO+4+4+18] = resolution;
        _packetBuffer[12+_cBBO+4+4+19] = interlaced;
        _packetBuffer[12+_cBBO+4+4+20] = 0x00; // YUV

        _finishCommandPacket();
}


    void ATEMext::setCameraControlLift(uint8_t input, int liftR, int lift
G, int liftB, int liftY) {
        _prepareCommandPacket(("CCmd"),24);
```

```
            // Preset values:
        _packetBuffer[12+_cBBO+4+4+1] = 8;
        _packetBuffer[12+_cBBO+4+4+2] = 0;

        _packetBuffer[12+_cBBO+4+4+4] = 0x80;
        _packetBuffer[12+_cBBO+4+4+9] = 0x04;

        _packetBuffer[12+_cBBO+4+4+0] = input;

        _packetBuffer[12+_cBBO+4+4+16] = highByte(liftR);
        _packetBuffer[12+_cBBO+4+4+17] = lowByte(liftR);
        _packetBuffer[12+_cBBO+4+4+18] = highByte(liftG);
        _packetBuffer[12+_cBBO+4+4+19] = lowByte(liftG);
        _packetBuffer[12+_cBBO+4+4+20] = highByte(liftB);
        _packetBuffer[12+_cBBO+4+4+21] = lowByte(liftB);
        _packetBuffer[12+_cBBO+4+4+22] = highByte(liftY);
        _packetBuffer[12+_cBBO+4+4+23] = lowByte(liftY);

        _finishCommandPacket();
    }

    void ATEMext::setCameraControlGamma(uint8_t input, int gammaR, int ga
mmaG, int gammaB, int gammaY) {
        _prepareCommandPacket(("CCmd"),24);

            // Preset values:
        _packetBuffer[12+_cBBO+4+4+1] = 8;
        _packetBuffer[12+_cBBO+4+4+2] = 1;

        _packetBuffer[12+_cBBO+4+4+4] = 0x80;
        _packetBuffer[12+_cBBO+4+4+9] = 0x04;

        _packetBuffer[12+_cBBO+4+4+0] = input;

        _packetBuffer[12+_cBBO+4+4+16] = highByte(gammaR);
        _packetBuffer[12+_cBBO+4+4+17] = lowByte(gammaR);
        _packetBuffer[12+_cBBO+4+4+18] = highByte(gammaG);
        _packetBuffer[12+_cBBO+4+4+19] = lowByte(gammaG);
        _packetBuffer[12+_cBBO+4+4+20] = highByte(gammaB);
        _packetBuffer[12+_cBBO+4+4+21] = lowByte(gammaB);
        _packetBuffer[12+_cBBO+4+4+22] = highByte(gammaY);
        _packetBuffer[12+_cBBO+4+4+23] = lowByte(gammaY);


        _finishCommandPacket();
    }

    void ATEMext::setCameraControlGain(uint8_t input, int gainR, int gain
G, int gainB, int gainY) {
```

```
        _prepareCommandPacket(("CCmd"),24);

            // Preset values:
        _packetBuffer[12+_cBBO+4+4+1] = 8;
        _packetBuffer[12+_cBBO+4+4+2] = 2;

        _packetBuffer[12+_cBBO+4+4+4] = 0x80;
        _packetBuffer[12+_cBBO+4+4+9] = 0x04;

        _packetBuffer[12+_cBBO+4+4+0] = input;

        _packetBuffer[12+_cBBO+4+4+16] = highByte(gainR);
        _packetBuffer[12+_cBBO+4+4+17] = lowByte(gainR);
        _packetBuffer[12+_cBBO+4+4+18] = highByte(gainG);
        _packetBuffer[12+_cBBO+4+4+19] = lowByte(gainG);
        _packetBuffer[12+_cBBO+4+4+20] = highByte(gainB);
        _packetBuffer[12+_cBBO+4+4+21] = lowByte(gainB);
        _packetBuffer[12+_cBBO+4+4+22] = highByte(gainY);
        _packetBuffer[12+_cBBO+4+4+23] = lowByte(gainY);

        _finishCommandPacket();
    }

    void ATEMext::setCameraControlHueSaturation(uint8_t input, int hue, i
nt saturation) {
        _prepareCommandPacket(("CCmd"),24);

            // Preset values:
        _packetBuffer[12+_cBBO+4+4+1] = 8;
        _packetBuffer[12+_cBBO+4+4+2] = 6;

        _packetBuffer[12+_cBBO+4+4+4] = 0x80;
        _packetBuffer[12+_cBBO+4+4+9] = 0x02;

        _packetBuffer[12+_cBBO+4+4+0] = input;

        _packetBuffer[12+_cBBO+4+4+16] = highByte(hue);
        _packetBuffer[12+_cBBO+4+4+17] = lowByte(hue);

        _packetBuffer[12+_cBBO+4+4+18] = highByte(saturation);
        _packetBuffer[12+_cBBO+4+4+19] = lowByte(saturation);

        _finishCommandPacket();
    }



        // *********************************
        // **
```

```
        // ** Implementations in ATEMext.c:
        // **
        // ********************************

        void ATEMext::_parseGetCommands(const char *cmdStr) {
            uint8_t mE,multiViewer,windowIndex,keyer,colorGenerator,aUXCh
annel,input,mediaPlayer,clipBank,macroIndex,box;
            uint16_t videoSource,index,audioSource,sources;
            long temp;
            uint8_t readBytesForTlSr;

            if (!strcmp(cmdStr, ("AMLv")))  {
                _readToPacketBuffer(36);
            } else if (!strcmp(cmdStr, ("TlSr")))   {
                readBytesForTlSr = ((ATEM_packetBufferLength-2)/3)*3+2;
                _readToPacketBuffer(readBytesForTlSr);
            } else {
                _readToPacketBuffer();  // Default
            }


            if (!strcmp(cmdStr, ("_pin")))  {
                if (_packetBuffer[5]=='T')  {
                        _ATEMmodel = 0;
                } else
                if (_packetBuffer[5]=='1')  {
                        _ATEMmodel = _packetBuffer[29]=='4' ? 4 : 1;
                } else
                if (_packetBuffer[5]=='2')  {
                    _ATEMmodel = _packetBuffer[29]=='4' ? 5 : 2;
                } else
                if (_packetBuffer[5]=='P')  {
                        _ATEMmodel = 3;
                }

                #if ATEM_debug
                if (_serialOutput>0)    {
                    Serial.print(F("Switcher type: "));
                    Serial.print(_ATEMmodel);
                    switch(_ATEMmodel)  {
                        case 0:
                            Serial.println(F(" - TeleVision Studio"));
                        break;
                        case 1:
                            Serial.println(F(" - ATEM 1 M/E"));
                        break;
                        case 2:
                            Serial.println(F(" - ATEM 2 M/E"));
                        break;
```

```
                        case 3:
                            Serial.println(F(" - ATEM Production Studio 4
K"));
                        break;
                        case 4:
                            Serial.println(F(" - ATEM 1 M/E 4K"));
                        break;
                        case 5:
                            Serial.println(F(" - ATEM 2 M/E 4K"));
                        break;
                    }
                }
                #endif
            }




            if(!strcmp(cmdStr, ("_ver"))) {

                    #if ATEM_debug
                    temp = atemProtocolVersionMajor;
                    #endif
                    atemProtocolVersionMajor = word(_packetBuffer[0], _pa
cketBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemProtocolVersionMajor!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemProtocolVersionMajor = "));
                        Serial.println(atemProtocolVersionMajor);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemProtocolVersionMinor;
                    #endif
                    atemProtocolVersionMinor = word(_packetBuffer[2], _pa
cketBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemProtocolVersionMinor!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemProtocolVersionMinor = "));
                        Serial.println(atemProtocolVersionMinor);
                    }
                    #endif

            } else
```

```
            if(!strcmp(cmdStr, ("_pin"))) {

                    memset(atemProductIdName,0,45);
                    strncpy(atemProductIdName, (char *)(_packetBuffer+0),
 44);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && hasInitialized()) || (_se
rialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemProductIdName = "));
                        Serial.println(atemProductIdName);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("_top"))) {

                    #if ATEM_debug
                    temp = atemTopologyMEs;
                    #endif
                    atemTopologyMEs = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTopologyMEs!=temp) ||
 (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTopologyMEs = "));
                        Serial.println(atemTopologyMEs);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTopologySources;
                    #endif
                    atemTopologySources = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTopologySources!=temp
) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTopologySources = "));
                        Serial.println(atemTopologySources);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTopologyColorGenerators;
                    #endif
                    atemTopologyColorGenerators = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTopologyColorGenerato
rs!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTopologyColorGenerators = "))
;
```

```
                        Serial.println(atemTopologyColorGenerators);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTopologyAUXbusses;
                    #endif
                    atemTopologyAUXbusses = _packetBuffer[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTopologyAUXbusses!=te
mp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTopologyAUXbusses = "));
                        Serial.println(atemTopologyAUXbusses);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTopologyDownstreamKeyes;
                    #endif
                    atemTopologyDownstreamKeyes = _packetBuffer[4];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTopologyDownstreamKey
es!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemTopologyDownstreamKeyes = "))
;
                        Serial.println(atemTopologyDownstreamKeyes);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTopologyStingers;
                    #endif
                    atemTopologyStingers = _packetBuffer[5];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTopologyStingers!=tem
p) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTopologyStingers = "));
                        Serial.println(atemTopologyStingers);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTopologyDVEs;
                    #endif
                    atemTopologyDVEs = _packetBuffer[6];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTopologyDVEs!=temp) |
| (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTopologyDVEs = "));
```

```
                    Serial.println(atemTopologyDVEs);
                }
                #endif

                #if ATEM_debug
                temp = atemTopologySuperSources;
                #endif
                atemTopologySuperSources = _packetBuffer[7];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTopologySuperSources!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                    Serial.print(F("atemTopologySuperSources = "));
                    Serial.println(atemTopologySuperSources);
                }
                #endif

                #if ATEM_debug
                temp = atemTopologyHasSDOutput;
                #endif
                atemTopologyHasSDOutput = _packetBuffer[9];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTopologyHasSDOutput!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                    Serial.print(F("atemTopologyHasSDOutput = "));
                    Serial.println(atemTopologyHasSDOutput);
                }
                #endif

        } else
        if(!strcmp(cmdStr, ("_MeC"))) {

            mE = _packetBuffer[0];
            if (mE<=1) {
                #if ATEM_debug
                temp = atemMixEffectConfigKeyersOnME[mE];
                #endif
                atemMixEffectConfigKeyersOnME[mE] = _packetBuffer[1];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemMixEffectConfigKeyers
OnME[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                    Serial.print(F("atemMixEffectConfigKeyersOnME[mE=
")); Serial.print(mE); Serial.print(F("] = "));
                    Serial.println(atemMixEffectConfigKeyersOnME[mE])
;
                }
                #endif

            }
        } else
```

```
            if(!strcmp(cmdStr, ("_mpl"))) {

                    #if ATEM_debug
                    temp = atemMediaPlayersStillBanks;
                    #endif
                    atemMediaPlayersStillBanks = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayersStillBank
s!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMediaPlayersStillBanks = "));
                        Serial.println(atemMediaPlayersStillBanks);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemMediaPlayersClipBanks;
                    #endif
                    atemMediaPlayersClipBanks = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayersClipBanks
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemMediaPlayersClipBanks = "));
                        Serial.println(atemMediaPlayersClipBanks);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("_MvC"))) {

                    #if ATEM_debug
                    temp = atemMultiViewConfigMultiViewers;
                    #endif
                    atemMultiViewConfigMultiViewers = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMultiViewConfigMultiV
iewers!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemMultiViewConfigMultiViewers =
 "));

                        Serial.println(atemMultiViewConfigMultiViewers);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("_SSC"))) {

                    #if ATEM_debug
                    temp = atemSuperSourceConfigBoxes;
                    #endif
                    atemSuperSourceConfigBoxes = _packetBuffer[0];
```

```
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceConfigBoxe
s!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemSuperSourceConfigBoxes = "));
                        Serial.println(atemSuperSourceConfigBoxes);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("_AMC"))) {

                    #if ATEM_debug
                    temp = atemAudioMixerConfigAudioChannels;
                    #endif
                    atemAudioMixerConfigAudioChannels = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerConfigAudio
Channels!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemAudioMixerConfigAudioChannels
 = "));

                        Serial.println(atemAudioMixerConfigAudioChannels)
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerConfigHasMonitor;
                    #endif
                    atemAudioMixerConfigHasMonitor = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerConfigHasMo
nitor!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemAudioMixerConfigHasMonitor =
"));

                        Serial.println(atemAudioMixerConfigHasMonitor);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("_VMC"))) {

                    #if ATEM_debug
                    temp = atemVideoMixerConfigModes;
                    #endif
                    atemVideoMixerConfigModes = (uint32_t)_packetBuffer[1
]<<16 | (uint32_t)_packetBuffer[2]<<8 | (uint32_t)_packetBuffer[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemVideoMixerConfigModes
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
```

```
                        Serial.print(F("atemVideoMixerConfigModes = "));
                        Serial.println(atemVideoMixerConfigModes);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("_MAC"))) {

                    #if ATEM_debug
                    temp = atemMacroPoolBanks;
                    #endif
                    atemMacroPoolBanks = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMacroPoolBanks!=temp)
 || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMacroPoolBanks = "));
                        Serial.println(atemMacroPoolBanks);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("DcOt"))) {

                    #if ATEM_debug
                    temp = atemDownConverterMode;
                    #endif
                    atemDownConverterMode = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownConverterMode!=te
mp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemDownConverterMode = "));
                        Serial.println(atemDownConverterMode);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("VidM"))) {

                    #if ATEM_debug
                    temp = atemVideoModeFormat;
                    #endif
                    atemVideoModeFormat = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemVideoModeFormat!=temp
) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemVideoModeFormat = "));
                        Serial.println(atemVideoModeFormat);
                    }
                    #endif
```

```
            } else
            if(!strcmp(cmdStr, ("InPr"))) {

                videoSource = word(_packetBuffer[0],_packetBuffer[1]);
                if (getVideoSrcIndex(videoSource)<=46) {
                    memset(atemInputShortName[getVideoSrcIndex(videoSourc
e)],0,5);
                    strncpy(atemInputShortName[getVideoSrcIndex(videoSour
ce)], (char *)(_packetBuffer+22), 4);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && hasInitialized()) || (_se
rialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemInputShortName[getVideoSrcInd
ex(videoSource)=")); Serial.print(getVideoSrcIndex(videoSource)); Serial.
print(F("] = "));
                        Serial.println(atemInputShortName[getVideoSrcInde
x(videoSource)]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemInputAvailability[getVideoSrcIndex(videoSo
urce)];
                    #endif
                    atemInputAvailability[getVideoSrcIndex(videoSource)]
= _packetBuffer[34];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemInputAvailability[get
VideoSrcIndex(videoSource)]!=temp) || (_serialOutput==0x81 && !hasInitial
ized()))  {
                        Serial.print(F("atemInputAvailability[getVideoSrc
Index(videoSource)=")); Serial.print(getVideoSrcIndex(videoSource)); Seri
al.print(F("] = "));
                        Serial.println(atemInputAvailability[getVideoSrcI
ndex(videoSource)]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemInputMEAvailability[getVideoSrcIndex(video
Source)];
                    #endif
                    atemInputMEAvailability[getVideoSrcIndex(videoSource)
] = _packetBuffer[35];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemInputMEAvailability[g
etVideoSrcIndex(videoSource)]!=temp) || (_serialOutput==0x81 && !hasIniti
alized()))    {
```

```
                        Serial.print(F("atemInputMEAvailability[getVideoS
rcIndex(videoSource)=")); Serial.print(getVideoSrcIndex(videoSource)); Se
rial.print(F("] = "));
                        Serial.println(atemInputMEAvailability[getVideoSr
cIndex(videoSource)]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("MvIn"))) {

                multiViewer = _packetBuffer[0];
                windowIndex = _packetBuffer[1];
                if (multiViewer<=1 && windowIndex<=9) {
                    #if ATEM_debug
                    temp = atemMultiViewerInputVideoSource[multiViewer][w
indowIndex];
                    #endif
                    atemMultiViewerInputVideoSource[multiViewer][windowIn
dex] = word(_packetBuffer[2], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMultiViewerInputVideo
Source[multiViewer][windowIndex]!=temp) || (_serialOutput==0x81 && !hasIn
itialized())) {
                        Serial.print(F("atemMultiViewerInputVideoSource[m
ultiViewer=")); Serial.print(multiViewer); Serial.print(F("][windowIndex=
")); Serial.print(windowIndex); Serial.print(F("] = "));
                        Serial.println(atemMultiViewerInputVideoSource[mu
ltiViewer][windowIndex]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("PrgI"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemProgramInputVideoSource[mE];
                    #endif
                    atemProgramInputVideoSource[mE] = word(_packetBuffer[
2], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemProgramInputVideoSour
ce[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemProgramInputVideoSource[mE=")
); Serial.print(mE); Serial.print(F("] = "));
```

```
                    Serial.println(atemProgramInputVideoSource[mE]);
                }
                #endif

            }
        } else
        if(!strcmp(cmdStr, ("PrvI"))) {

            mE = _packetBuffer[0];
            if (mE<=1) {
                #if ATEM_debug
                temp = atemPreviewInputVideoSource[mE];
                #endif
                atemPreviewInputVideoSource[mE] = word(_packetBuffer[
2], _packetBuffer[3]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemPreviewInputVideoSour
ce[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                    Serial.print(F("atemPreviewInputVideoSource[mE=")
); Serial.print(mE); Serial.print(F("] = "));
                    Serial.println(atemPreviewInputVideoSource[mE]);
                }
                #endif

            }
        } else
        if(!strcmp(cmdStr, ("TrSS"))) {

            mE = _packetBuffer[0];
            if (mE<=1) {
                #if ATEM_debug
                temp = atemTransitionStyle[mE];
                #endif
                atemTransitionStyle[mE] = _packetBuffer[1];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTransitionStyle[mE]!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                    Serial.print(F("atemTransitionStyle[mE=")); Seria
l.print(mE); Serial.print(F("] = "));
                    Serial.println(atemTransitionStyle[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionNextTransition[mE];
                #endif
                atemTransitionNextTransition[mE] = _packetBuffer[2];
                #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemTransitionNextTransit
ion[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTransitionNextTransition[mE="
)); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionNextTransition[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("TrPr"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionPreviewEnabled[mE];
                    #endif
                    atemTransitionPreviewEnabled[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionPreviewEnab
led[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTransitionPreviewEnabled[mE="
)); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionPreviewEnabled[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("TrPs"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionInTransition[mE];
                    #endif
                    atemTransitionInTransition[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionInTransitio
n[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionInTransition[mE="))
; Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionInTransition[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionFramesRemaining[mE];
                    #endif
```

```
                    atemTransitionFramesRemaining[mE] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionFramesRemai
ning[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionFramesRemaining[mE=
")); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionFramesRemaining[mE])
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionPosition[mE];
                    #endif
                    atemTransitionPosition[mE] = word(_packetBuffer[4], _
packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionPosition[mE
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionPosition[mE=")); Se
rial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionPosition[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("TMxP"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionMixRate[mE];
                    #endif
                    atemTransitionMixRate[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionMixRate[mE]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionMixRate[mE=")); Ser
ial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionMixRate[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("TDpP"))) {

                mE = _packetBuffer[0];
```

```
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionDipRate[mE];
                    #endif
                    atemTransitionDipRate[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDipRate[mE]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionDipRate[mE=")); Ser
ial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDipRate[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDipInput[mE];
                    #endif
                    atemTransitionDipInput[mE] = word(_packetBuffer[2], _
packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDipInput[mE
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionDipInput[mE=")); Se
rial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDipInput[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("TWpP"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionWipeRate[mE];
                    #endif
                    atemTransitionWipeRate[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionWipeRate[mE
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionWipeRate[mE=")); Se
rial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionWipeRate[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionWipePattern[mE];
```

```
                    #endif
                    atemTransitionWipePattern[mE] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionWipePattern
[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionWipePattern[mE="));
 Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionWipePattern[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionWipeWidth[mE];
                    #endif
                    atemTransitionWipeWidth[mE] = word(_packetBuffer[4],
_packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionWipeWidth[m
E]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemTransitionWipeWidth[mE=")); S
erial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionWipeWidth[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionWipeFillSource[mE];
                    #endif
                    atemTransitionWipeFillSource[mE] = word(_packetBuffer
[6], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionWipeFillSou
rce[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTransitionWipeFillSource[mE="
)); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionWipeFillSource[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionWipeSymmetry[mE];
                    #endif
                    atemTransitionWipeSymmetry[mE] = word(_packetBuffer[8
], _packetBuffer[9]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionWipeSymmetr
y[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionWipeSymmetry[mE="))
; Serial.print(mE); Serial.print(F("] = "));
```

```
                    Serial.println(atemTransitionWipeSymmetry[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionWipeSoftness[mE];
                #endif
                atemTransitionWipeSoftness[mE] = word(_packetBuffer[1
0], _packetBuffer[11]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTransitionWipeSoftnes
s[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                    Serial.print(F("atemTransitionWipeSoftness[mE="))
; Serial.print(mE); Serial.print(F("] = "));
                    Serial.println(atemTransitionWipeSoftness[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionWipePositionX[mE];
                #endif
                atemTransitionWipePositionX[mE] = word(_packetBuffer[
12], _packetBuffer[13]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTransitionWipePositio
nX[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                    Serial.print(F("atemTransitionWipePositionX[mE=")
); Serial.print(mE); Serial.print(F("] = "));
                    Serial.println(atemTransitionWipePositionX[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionWipePositionY[mE];
                #endif
                atemTransitionWipePositionY[mE] = word(_packetBuffer[
14], _packetBuffer[15]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTransitionWipePositio
nY[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                    Serial.print(F("atemTransitionWipePositionY[mE=")
); Serial.print(mE); Serial.print(F("] = "));
                    Serial.println(atemTransitionWipePositionY[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionWipeReverse[mE];
                #endif
```

```
                    atemTransitionWipeReverse[mE] = _packetBuffer[16];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionWipeReverse
[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionWipeReverse[mE="));
 Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionWipeReverse[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionWipeFlipFlop[mE];
                    #endif
                    atemTransitionWipeFlipFlop[mE] = _packetBuffer[17];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionWipeFlipFlo
p[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionWipeFlipFlop[mE="))
; Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionWipeFlipFlop[mE]);
                    }
                    #endif

      }

            } else
            if(!strcmp(cmdStr, ("TDvP"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionDVERate[mE];
                    #endif
                    atemTransitionDVERate[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVERate[mE]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionDVERate[mE=")); Ser
ial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVERate[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEStyle[mE];
                    #endif
                    atemTransitionDVEStyle[mE] = _packetBuffer[3];
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemTransitionDVEStyle[mE
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionDVEStyle[mE=")); Se
rial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEStyle[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEFillSource[mE];
                    #endif
                    atemTransitionDVEFillSource[mE] = word(_packetBuffer[
4], _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVEFillSour
ce[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionDVEFillSource[mE=")
); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEFillSource[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEKeySource[mE];
                    #endif
                    atemTransitionDVEKeySource[mE] = word(_packetBuffer[6
], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVEKeySourc
e[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionDVEKeySource[mE="))
; Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEKeySource[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEEnableKey[mE];
                    #endif
                    atemTransitionDVEEnableKey[mE] = _packetBuffer[8];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVEEnableKe
y[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionDVEEnableKey[mE="))
; Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEEnableKey[mE]);
                    }
                    #endif
```

```
                    #if ATEM_debug
                    temp = atemTransitionDVEPreMultiplied[mE];
                    #endif
                    atemTransitionDVEPreMultiplied[mE] = _packetBuffer[9]
;
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVEPreMulti
plied[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionDVEPreMultiplied[mE
=")); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEPreMultiplied[mE]
);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEClip[mE];
                    #endif
                    atemTransitionDVEClip[mE] = word(_packetBuffer[10], _
packetBuffer[11]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVEClip[mE]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionDVEClip[mE=")); Ser
ial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEClip[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEGain[mE];
                    #endif
                    atemTransitionDVEGain[mE] = word(_packetBuffer[12], _
packetBuffer[13]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVEGain[mE]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionDVEGain[mE=")); Ser
ial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEGain[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEInvertKey[mE];
                    #endif
                    atemTransitionDVEInvertKey[mE] = _packetBuffer[14];
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemTransitionDVEInvertKe
y[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionDVEInvertKey[mE="))
; Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEInvertKey[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEReverse[mE];
                    #endif
                    atemTransitionDVEReverse[mE] = _packetBuffer[15];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVEReverse[
mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTransitionDVEReverse[mE="));
Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEReverse[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionDVEFlipFlop[mE];
                    #endif
                    atemTransitionDVEFlipFlop[mE] = _packetBuffer[16];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionDVEFlipFlop
[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionDVEFlipFlop[mE="));
 Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionDVEFlipFlop[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("TStP"))) {

                mE = _packetBuffer[0];
                if (mE<=1) {
                    #if ATEM_debug
                    temp = atemTransitionStingerSource[mE];
                    #endif
                    atemTransitionStingerSource[mE] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionStingerSour
ce[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionStingerSource[mE=")
); Serial.print(mE); Serial.print(F("] = "));
```

```
                    Serial.println(atemTransitionStingerSource[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionStingerPreMultiplied[mE];
                #endif
                atemTransitionStingerPreMultiplied[mE] = _packetBuffe
r[2];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTransitionStingerPreM
ultiplied[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                    Serial.print(F("atemTransitionStingerPreMultiplie
d[mE=")); Serial.print(mE); Serial.print(F("] = "));
                    Serial.println(atemTransitionStingerPreMultiplied
[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionStingerClip[mE];
                #endif
                atemTransitionStingerClip[mE] = word(_packetBuffer[4]
, _packetBuffer[5]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTransitionStingerClip
[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                    Serial.print(F("atemTransitionStingerClip[mE="));
 Serial.print(mE); Serial.print(F("] = "));
                    Serial.println(atemTransitionStingerClip[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionStingerGain[mE];
                #endif
                atemTransitionStingerGain[mE] = word(_packetBuffer[6]
, _packetBuffer[7]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemTransitionStingerGain
[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                    Serial.print(F("atemTransitionStingerGain[mE="));
 Serial.print(mE); Serial.print(F("] = "));
                    Serial.println(atemTransitionStingerGain[mE]);
                }
                #endif

                #if ATEM_debug
                temp = atemTransitionStingerInvertKey[mE];
```

```
                    #endif
                    atemTransitionStingerInvertKey[mE] = _packetBuffer[8]
;
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionStingerInve
rtKey[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemTransitionStingerInvertKey[mE
=")); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionStingerInvertKey[mE]
);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionStingerPreRoll[mE];
                    #endif
                    atemTransitionStingerPreRoll[mE] = word(_packetBuffer
[10], _packetBuffer[11]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionStingerPreR
oll[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTransitionStingerPreRoll[mE="
)); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionStingerPreRoll[mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionStingerClipDuration[mE];
                    #endif
                    atemTransitionStingerClipDuration[mE] = word(_packetB
uffer[12], _packetBuffer[13]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionStingerClip
Duration[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionStingerClipDuration
[mE=")); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionStingerClipDuration[
mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionStingerTriggerPoint[mE];
                    #endif
                    atemTransitionStingerTriggerPoint[mE] = word(_packetB
uffer[14], _packetBuffer[15]);
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemTransitionStingerTrig
gerPoint[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemTransitionStingerTriggerPoint
[mE=")); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionStingerTriggerPoint[
mE]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemTransitionStingerMixRate[mE];
                    #endif
                    atemTransitionStingerMixRate[mE] = word(_packetBuffer
[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTransitionStingerMixR
ate[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTransitionStingerMixRate[mE="
)); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemTransitionStingerMixRate[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("KeOn"))) {

                mE = _packetBuffer[0];
                keyer = _packetBuffer[1];
                if (mE<=1 && keyer<=3) {
                    #if ATEM_debug
                    temp = atemKeyerOnAirEnabled[mE][keyer];
                    #endif
                    atemKeyerOnAirEnabled[mE][keyer] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerOnAirEnabled[mE]
[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyerOnAirEnabled[mE=")); Ser
ial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.p
rint(F("] = "));
                        Serial.println(atemKeyerOnAirEnabled[mE][keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("KeBP"))) {

                mE = _packetBuffer[0];
```

```
                keyer = _packetBuffer[1];
                if (mE<=1 && keyer<=3) {
                    #if ATEM_debug
                    temp = atemKeyerType[mE][keyer];
                    #endif
                    atemKeyerType[mE][keyer] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerType[mE][keyer]!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyerType[mE=")); Serial.prin
t(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("
] = "));
                        Serial.println(atemKeyerType[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyerFlyEnabled[mE][keyer];
                    #endif
                    atemKeyerFlyEnabled[mE][keyer] = _packetBuffer[5];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerFlyEnabled[mE][k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyerFlyEnabled[mE=")); Seria
l.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.pri
nt(F("] = "));
                        Serial.println(atemKeyerFlyEnabled[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyerFillSource[mE][keyer];
                    #endif
                    atemKeyerFillSource[mE][keyer] = word(_packetBuffer[6
], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerFillSource[mE][k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyerFillSource[mE=")); Seria
l.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.pri
nt(F("] = "));
                        Serial.println(atemKeyerFillSource[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyerKeySource[mE][keyer];
                    #endif
```

```
                    atemKeyerKeySource[mE][keyer] = word(_packetBuffer[8]
, _packetBuffer[9]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerKeySource[mE][ke
yer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyerKeySource[mE=")); Serial
.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.prin
t(F("] = "));
                        Serial.println(atemKeyerKeySource[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyerMasked[mE][keyer];
                    #endif
                    atemKeyerMasked[mE][keyer] = _packetBuffer[10];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerMasked[mE][keyer
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyerMasked[mE=")); Serial.pr
int(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F
("] = "));
                        Serial.println(atemKeyerMasked[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyerTop[mE][keyer];
                    #endif
                    atemKeyerTop[mE][keyer] = (int16_t) word(_packetBuffe
r[12], _packetBuffer[13]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerTop[mE][keyer]!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemKeyerTop[mE=")); Serial.print
(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("]
 = "));
                        Serial.println(atemKeyerTop[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyerBottom[mE][keyer];
                    #endif
                    atemKeyerBottom[mE][keyer] = (int16_t) word(_packetBu
ffer[14], _packetBuffer[15]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerBottom[mE][keyer
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
```

```
                        Serial.print(F("atemKeyerBottom[mE=")); Serial.pr
int(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F
("] = "));
                        Serial.println(atemKeyerBottom[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyerLeft[mE][keyer];
                    #endif
                    atemKeyerLeft[mE][keyer] = (int16_t) word(_packetBuff
er[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerLeft[mE][keyer]!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyerLeft[mE=")); Serial.prin
t(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("
] = "));
                        Serial.println(atemKeyerLeft[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyerRight[mE][keyer];
                    #endif
                    atemKeyerRight[mE][keyer] = (int16_t) word(_packetBuf
fer[18], _packetBuffer[19]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyerRight[mE][keyer]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyerRight[mE=")); Serial.pri
nt(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F(
"] = "));
                        Serial.println(atemKeyerRight[mE][keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("KeLm"))) {

                mE = _packetBuffer[0];
                keyer = _packetBuffer[1];
                if (mE<=1 && keyer<=3) {
                    #if ATEM_debug
                    temp = atemKeyLumaPreMultiplied[mE][keyer];
                    #endif
                    atemKeyLumaPreMultiplied[mE][keyer] = _packetBuffer[2
];
```

```
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyLumaPreMultiplied[
mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemKeyLumaPreMultiplied[mE="));
Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Seria
l.print(F("] = "));
                        Serial.println(atemKeyLumaPreMultiplied[mE][keyer
]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyLumaClip[mE][keyer];
                    #endif
                    atemKeyLumaClip[mE][keyer] = word(_packetBuffer[4], _
packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyLumaClip[mE][keyer
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyLumaClip[mE=")); Serial.pr
int(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F
("] = "));
                        Serial.println(atemKeyLumaClip[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyLumaGain[mE][keyer];
                    #endif
                    atemKeyLumaGain[mE][keyer] = word(_packetBuffer[6], _
packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyLumaGain[mE][keyer
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyLumaGain[mE=")); Serial.pr
int(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F
("] = "));
                        Serial.println(atemKeyLumaGain[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyLumaInvertKey[mE][keyer];
                    #endif
                    atemKeyLumaInvertKey[mE][keyer] = _packetBuffer[8];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyLumaInvertKey[mE][
keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
```

```
                        Serial.print(F("atemKeyLumaInvertKey[mE=")); Seri
al.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.pr
int(F("] = "));
                        Serial.println(atemKeyLumaInvertKey[mE][keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("KeCk"))) {

                mE = _packetBuffer[0];
                keyer = _packetBuffer[1];
                if (mE<=1 && keyer<=3) {
                    #if ATEM_debug
                    temp = atemKeyChromaHue[mE][keyer];
                    #endif
                    atemKeyChromaHue[mE][keyer] = word(_packetBuffer[2],
_packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyChromaHue[mE][keye
r]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemKeyChromaHue[mE=")); Serial.p
rint(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(
F("] = "));
                        Serial.println(atemKeyChromaHue[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyChromaGain[mE][keyer];
                    #endif
                    atemKeyChromaGain[mE][keyer] = word(_packetBuffer[4],
 _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyChromaGain[mE][key
er]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyChromaGain[mE=")); Serial.
print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print
(F("] = "));
                        Serial.println(atemKeyChromaGain[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyChromaYSuppress[mE][keyer];
                    #endif
                    atemKeyChromaYSuppress[mE][keyer] = word(_packetBuffe
r[6], _packetBuffer[7]);
```

```
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyChromaYSuppress[mE
][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyChromaYSuppress[mE=")); Se
rial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.
print(F("] = "));
                        Serial.println(atemKeyChromaYSuppress[mE][keyer])
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyChromaLift[mE][keyer];
                    #endif
                    atemKeyChromaLift[mE][keyer] = word(_packetBuffer[8],
 _packetBuffer[9]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyChromaLift[mE][key
er]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyChromaLift[mE=")); Serial.
print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print
(F("] = "));
                        Serial.println(atemKeyChromaLift[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyChromaNarrow[mE][keyer];
                    #endif
                    atemKeyChromaNarrow[mE][keyer] = _packetBuffer[10];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyChromaNarrow[mE][k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyChromaNarrow[mE=")); Seria
l.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.pri
nt(F("] = "));
                        Serial.println(atemKeyChromaNarrow[mE][keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("KePt"))) {

                mE = _packetBuffer[0];
                keyer = _packetBuffer[1];
                if (mE<=1 && keyer<=3) {
                    #if ATEM_debug
                    temp = atemKeyPatternPattern[mE][keyer];
```

```
                    #endif
                    atemKeyPatternPattern[mE][keyer] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyPatternPattern[mE]
[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyPatternPattern[mE=")); Ser
ial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.p
rint(F("] = "));
                        Serial.println(atemKeyPatternPattern[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyPatternSize[mE][keyer];
                    #endif
                    atemKeyPatternSize[mE][keyer] = word(_packetBuffer[4]
, _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyPatternSize[mE][ke
yer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyPatternSize[mE=")); Serial
.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.prin
t(F("] = "));
                        Serial.println(atemKeyPatternSize[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyPatternSymmetry[mE][keyer];
                    #endif
                    atemKeyPatternSymmetry[mE][keyer] = word(_packetBuffe
r[6], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyPatternSymmetry[mE
][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyPatternSymmetry[mE=")); Se
rial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.
print(F("] = "));
                        Serial.println(atemKeyPatternSymmetry[mE][keyer])
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyPatternSoftness[mE][keyer];
                    #endif
                    atemKeyPatternSoftness[mE][keyer] = word(_packetBuffe
r[8], _packetBuffer[9]);
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemKeyPatternSoftness[mE
][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyPatternSoftness[mE=")); Se
rial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.
print(F("] = "));
                        Serial.println(atemKeyPatternSoftness[mE][keyer])
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyPatternPositionX[mE][keyer];
                    #endif
                    atemKeyPatternPositionX[mE][keyer] = word(_packetBuff
er[10], _packetBuffer[11]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyPatternPositionX[m
E][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyPatternPositionX[mE=")); S
erial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial
.print(F("] = "));
                        Serial.println(atemKeyPatternPositionX[mE][keyer]
);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyPatternPositionY[mE][keyer];
                    #endif
                    atemKeyPatternPositionY[mE][keyer] = word(_packetBuff
er[12], _packetBuffer[13]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyPatternPositionY[m
E][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyPatternPositionY[mE=")); S
erial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial
.print(F("] = "));
                        Serial.println(atemKeyPatternPositionY[mE][keyer]
);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyPatternInvertPattern[mE][keyer];
                    #endif
                    atemKeyPatternInvertPattern[mE][keyer] = _packetBuffe
r[14];
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemKeyPatternInvertPatte
rn[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyPatternInvertPattern[mE=")
); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Se
rial.print(F("] = "));
                        Serial.println(atemKeyPatternInvertPattern[mE][ke
yer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("KeDV"))) {

                mE = _packetBuffer[0];
                keyer = _packetBuffer[1];
                if (mE<=1 && keyer<=3) {
                    #if ATEM_debug
                    temp = atemKeyDVESizeX[mE][keyer];
                    #endif
                    atemKeyDVESizeX[mE][keyer] = (uint32_t)_packetBuffer[
4]<<24 | (uint32_t)_packetBuffer[5]<<16 | (uint32_t)_packetBuffer[6]<<8 |
 (uint32_t)_packetBuffer[7];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVESizeX[mE][keyer
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyDVESizeX[mE=")); Serial.pr
int(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F
("] = "));
                        Serial.println(atemKeyDVESizeX[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVESizeY[mE][keyer];
                    #endif
                    atemKeyDVESizeY[mE][keyer] = (uint32_t)_packetBuffer[
8]<<24 | (uint32_t)_packetBuffer[9]<<16 | (uint32_t)_packetBuffer[10]<<8
| (uint32_t)_packetBuffer[11];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVESizeY[mE][keyer
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyDVESizeY[mE=")); Serial.pr
int(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F
("] = "));
                        Serial.println(atemKeyDVESizeY[mE][keyer]);
                    }
                    #endif
```

```
				#if ATEM_debug
				temp = atemKeyDVEPositionX[mE][keyer];
				#endif
				atemKeyDVEPositionX[mE][keyer] = (uint32_t)_packetBuffer[12]<<24 | (uint32_t)_packetBuffer[13]<<16 | (uint32_t)_packetBuffer[14]<<8 | (uint32_t)_packetBuffer[15];
				#if ATEM_debug
				if ((_serialOutput==0x80 && atemKeyDVEPositionX[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))	{
					Serial.print(F("atemKeyDVEPositionX[mE=")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("] = "));
					Serial.println(atemKeyDVEPositionX[mE][keyer]);
				}
				#endif

				#if ATEM_debug
				temp = atemKeyDVEPositionY[mE][keyer];
				#endif
				atemKeyDVEPositionY[mE][keyer] = (uint32_t)_packetBuffer[16]<<24 | (uint32_t)_packetBuffer[17]<<16 | (uint32_t)_packetBuffer[18]<<8 | (uint32_t)_packetBuffer[19];
				#if ATEM_debug
				if ((_serialOutput==0x80 && atemKeyDVEPositionY[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))	{
					Serial.print(F("atemKeyDVEPositionY[mE=")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("] = "));
					Serial.println(atemKeyDVEPositionY[mE][keyer]);
				}
				#endif

				#if ATEM_debug
				temp = atemKeyDVERotation[mE][keyer];
				#endif
				atemKeyDVERotation[mE][keyer] = (uint32_t)_packetBuffer[20]<<24 | (uint32_t)_packetBuffer[21]<<16 | (uint32_t)_packetBuffer[22]<<8 | (uint32_t)_packetBuffer[23];
				#if ATEM_debug
				if ((_serialOutput==0x80 && atemKeyDVERotation[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
					Serial.print(F("atemKeyDVERotation[mE=")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("] = "));
					Serial.println(atemKeyDVERotation[mE][keyer]);
				}
				#endif

				#if ATEM_debug
```

```
                            temp = atemKeyDVEBorderEnabled[mE][keyer];
                            #endif
                            atemKeyDVEBorderEnabled[mE][keyer] = _packetBuffer[24
];
                            #if ATEM_debug
                            if ((_serialOutput==0x80 && atemKeyDVEBorderEnabled[m
E][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                                 Serial.print(F("atemKeyDVEBorderEnabled[mE=")); S
erial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial
.print(F("] = "));
                                 Serial.println(atemKeyDVEBorderEnabled[mE][keyer]
);
                            }
                            #endif

                            #if ATEM_debug
                            temp = atemKeyDVEShadow[mE][keyer];
                            #endif
                            atemKeyDVEShadow[mE][keyer] = _packetBuffer[25];
                            #if ATEM_debug
                            if ((_serialOutput==0x80 && atemKeyDVEShadow[mE][keye
r]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                                 Serial.print(F("atemKeyDVEShadow[mE=")); Serial.p
rint(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(
F("] = "));
                                 Serial.println(atemKeyDVEShadow[mE][keyer]);
                            }
                            #endif

                            #if ATEM_debug
                            temp = atemKeyDVEBorderBevel[mE][keyer];
                            #endif
                            atemKeyDVEBorderBevel[mE][keyer] = _packetBuffer[26];
                            #if ATEM_debug
                            if ((_serialOutput==0x80 && atemKeyDVEBorderBevel[mE]
[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                                 Serial.print(F("atemKeyDVEBorderBevel[mE=")); Ser
ial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.p
rint(F("] = "));
                                 Serial.println(atemKeyDVEBorderBevel[mE][keyer]);
                            }
                            #endif

                            #if ATEM_debug
                            temp = atemKeyDVEBorderOuterWidth[mE][keyer];
                            #endif
                            atemKeyDVEBorderOuterWidth[mE][keyer] = word(_packetB
uffer[28], _packetBuffer[29]);
                            #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemKeyDVEBorderOuterWidt
h[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyDVEBorderOuterWidth[mE="))
; Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Ser
ial.print(F("] = "));
                        Serial.println(atemKeyDVEBorderOuterWidth[mE][key
er]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderInnerWidth[mE][keyer];
                    #endif
                    atemKeyDVEBorderInnerWidth[mE][keyer] = word(_packetB
uffer[30], _packetBuffer[31]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVEBorderInnerWidt
h[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyDVEBorderInnerWidth[mE="))
; Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Ser
ial.print(F("] = "));
                        Serial.println(atemKeyDVEBorderInnerWidth[mE][key
er]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderOuterSoftness[mE][keyer];
                    #endif
                    atemKeyDVEBorderOuterSoftness[mE][keyer] = _packetBuf
fer[32];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVEBorderOuterSoft
ness[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyDVEBorderOuterSoftness[mE=
")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer);
Serial.print(F("] = "));
                        Serial.println(atemKeyDVEBorderOuterSoftness[mE][
keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderInnerSoftness[mE][keyer];
                    #endif
                    atemKeyDVEBorderInnerSoftness[mE][keyer] = _packetBuf
fer[33];
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemKeyDVEBorderInnerSoft
ness[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyDVEBorderInnerSoftness[mE=
")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer);
Serial.print(F("] = "));
                        Serial.println(atemKeyDVEBorderInnerSoftness[mE][
keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderBevelSoftness[mE][keyer];
                    #endif
                    atemKeyDVEBorderBevelSoftness[mE][keyer] = _packetBuf
fer[34];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVEBorderBevelSoft
ness[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyDVEBorderBevelSoftness[mE=
")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer);
Serial.print(F("] = "));
                        Serial.println(atemKeyDVEBorderBevelSoftness[mE][
keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderBevelPosition[mE][keyer];
                    #endif
                    atemKeyDVEBorderBevelPosition[mE][keyer] = _packetBuf
fer[35];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVEBorderBevelPosi
tion[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemKeyDVEBorderBevelPosition[mE=
")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer);
Serial.print(F("] = "));
                        Serial.println(atemKeyDVEBorderBevelPosition[mE][
keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderOpacity[mE][keyer];
                    #endif
                    atemKeyDVEBorderOpacity[mE][keyer] = _packetBuffer[36
];
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemKeyDVEBorderOpacity[m
E][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyDVEBorderOpacity[mE=")); S
erial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial
.print(F("] = "));
                        Serial.println(atemKeyDVEBorderOpacity[mE][keyer]
);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderHue[mE][keyer];
                    #endif
                    atemKeyDVEBorderHue[mE][keyer] = word(_packetBuffer[3
8], _packetBuffer[39]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVEBorderHue[mE][k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyDVEBorderHue[mE=")); Seria
l.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.pri
nt(F("] = "));
                        Serial.println(atemKeyDVEBorderHue[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderSaturation[mE][keyer];
                    #endif
                    atemKeyDVEBorderSaturation[mE][keyer] = word(_packetB
uffer[40], _packetBuffer[41]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVEBorderSaturatio
n[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyDVEBorderSaturation[mE="))
; Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Ser
ial.print(F("] = "));
                        Serial.println(atemKeyDVEBorderSaturation[mE][key
er]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVEBorderLuma[mE][keyer];
                    #endif
                    atemKeyDVEBorderLuma[mE][keyer] = word(_packetBuffer[
42], _packetBuffer[43]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVEBorderLuma[mE][
keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
```

```
                    Serial.print(F("atemKeyDVEBorderLuma[mE=")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                    Serial.println(atemKeyDVEBorderLuma[mE][keyer]);
                }
                #endif

                #if ATEM_debug
                temp = atemKeyDVELightSourceDirection[mE][keyer];
                #endif
                atemKeyDVELightSourceDirection[mE][keyer] = word(_packetBuffer[44], _packetBuffer[45]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemKeyDVELightSourceDirection[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                    Serial.print(F("atemKeyDVELightSourceDirection[mE=")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                    Serial.println(atemKeyDVELightSourceDirection[mE][keyer]);
                }
                #endif

                #if ATEM_debug
                temp = atemKeyDVELightSourceAltitude[mE][keyer];
                #endif
                atemKeyDVELightSourceAltitude[mE][keyer] = _packetBuffer[46];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemKeyDVELightSourceAltitude[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                    Serial.print(F("atemKeyDVELightSourceAltitude[mE=")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                    Serial.println(atemKeyDVELightSourceAltitude[mE][keyer]);
                }
                #endif

                #if ATEM_debug
                temp = atemKeyDVEMasked[mE][keyer];
                #endif
                atemKeyDVEMasked[mE][keyer] = _packetBuffer[47];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemKeyDVEMasked[mE][keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                    Serial.print(F("atemKeyDVEMasked[mE=")); Serial.print(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("] = "));
```

```
        Serial.println(atemKeyDVEMasked[mE][keyer]);
    }
    #endif

    #if ATEM_debug
    temp = atemKeyDVETop[mE][keyer];
    #endif
    atemKeyDVETop[mE][keyer] = (int16_t) word(_packetBuff
er[48], _packetBuffer[49]);
    #if ATEM_debug
    if ((_serialOutput==0x80 && atemKeyDVETop[mE][keyer]!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
        Serial.print(F("atemKeyDVETop[mE=")); Serial.prin
t(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F("
] = "));
        Serial.println(atemKeyDVETop[mE][keyer]);
    }
    #endif

    #if ATEM_debug
    temp = atemKeyDVEBottom[mE][keyer];
    #endif
    atemKeyDVEBottom[mE][keyer] = (int16_t) word(_packetB
uffer[50], _packetBuffer[51]);
    #if ATEM_debug
    if ((_serialOutput==0x80 && atemKeyDVEBottom[mE][keye
r]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
        Serial.print(F("atemKeyDVEBottom[mE=")); Serial.p
rint(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(
F("] = "));
        Serial.println(atemKeyDVEBottom[mE][keyer]);
    }
    #endif

    #if ATEM_debug
    temp = atemKeyDVELeft[mE][keyer];
    #endif
    atemKeyDVELeft[mE][keyer] = (int16_t) word(_packetBuf
fer[52], _packetBuffer[53]);
    #if ATEM_debug
    if ((_serialOutput==0x80 && atemKeyDVELeft[mE][keyer]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
        Serial.print(F("atemKeyDVELeft[mE=")); Serial.pri
nt(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F(
"] = "));
        Serial.println(atemKeyDVELeft[mE][keyer]);
    }
    #endif
```

```
                    #if ATEM_debug
                    temp = atemKeyDVERight[mE][keyer];
                    #endif
                    atemKeyDVERight[mE][keyer] = (int16_t) word(_packetBu
ffer[54], _packetBuffer[55]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVERight[mE][keyer
]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemKeyDVERight[mE=")); Serial.pr
int(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F
("] = "));
                        Serial.println(atemKeyDVERight[mE][keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemKeyDVERate[mE][keyer];
                    #endif
                    atemKeyDVERate[mE][keyer] = _packetBuffer[56];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemKeyDVERate[mE][keyer]
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemKeyDVERate[mE=")); Serial.pri
nt(mE); Serial.print(F("][keyer=")); Serial.print(keyer); Serial.print(F(
"] = "));
                        Serial.println(atemKeyDVERate[mE][keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("DskB"))) {

                keyer = _packetBuffer[0];
                if (keyer<=1) {
                    #if ATEM_debug
                    temp = atemDownstreamKeyerFillSource[keyer];
                    #endif
                    atemDownstreamKeyerFillSource[keyer] = word(_packetBu
ffer[2], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerFillSo
urce[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemDownstreamKeyerFillSource[key
er=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerFillSource[keye
r]);
                    }
                    #endif
```

```
                    #if ATEM_debug
                    temp = atemDownstreamKeyerKeySource[keyer];
                    #endif
                    atemDownstreamKeyerKeySource[keyer] = word(_packetBuf
fer[4], _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerKeySou
rce[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemDownstreamKeyerKeySource[keye
r=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerKeySource[keyer
]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("DskP"))) {

                keyer = _packetBuffer[0];
                if (keyer<=1) {
                    #if ATEM_debug
                    temp = atemDownstreamKeyerTie[keyer];
                    #endif
                    atemDownstreamKeyerTie[keyer] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerTie[ke
yer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemDownstreamKeyerTie[keyer="));
 Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerTie[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerRate[keyer];
                    #endif
                    atemDownstreamKeyerRate[keyer] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerRate[k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerRate[keyer="))
; Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerRate[keyer]);
                    }
                    #endif

                    #if ATEM_debug
```

```
                    temp = atemDownstreamKeyerPreMultiplied[keyer];
                    #endif
                    atemDownstreamKeyerPreMultiplied[keyer] = _packetBuff
er[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerPreMul
tiplied[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemDownstreamKeyerPreMultiplied[
keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerPreMultiplied[k
eyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerClip[keyer];
                    #endif
                    atemDownstreamKeyerClip[keyer] = word(_packetBuffer[4
], _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerClip[k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerClip[keyer="))
; Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerClip[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerGain[keyer];
                    #endif
                    atemDownstreamKeyerGain[keyer] = word(_packetBuffer[6
], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerGain[k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerGain[keyer="))
; Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerGain[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerInvertKey[keyer];
                    #endif
                    atemDownstreamKeyerInvertKey[keyer] = _packetBuffer[8
];
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemDownstreamKeyerInvert
Key[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemDownstreamKeyerInvertKey[keye
r=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerInvertKey[keyer
]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerMasked[keyer];
                    #endif
                    atemDownstreamKeyerMasked[keyer] = _packetBuffer[9];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerMasked
[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemDownstreamKeyerMasked[keyer="
)); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerMasked[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerTop[keyer];
                    #endif
                    atemDownstreamKeyerTop[keyer] = (int16_t) word(_packe
tBuffer[10], _packetBuffer[11]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerTop[ke
yer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemDownstreamKeyerTop[keyer="));
 Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerTop[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerBottom[keyer];
                    #endif
                    atemDownstreamKeyerBottom[keyer] = (int16_t) word(_pa
cketBuffer[12], _packetBuffer[13]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerBottom
[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemDownstreamKeyerBottom[keyer="
)); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerBottom[keyer]);
                    }
                    #endif
```

```
                    #if ATEM_debug
                    temp = atemDownstreamKeyerLeft[keyer];
                    #endif
                    atemDownstreamKeyerLeft[keyer] = (int16_t) word(_pack
etBuffer[14], _packetBuffer[15]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerLeft[k
eyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerLeft[keyer="))
; Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerLeft[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerRight[keyer];
                    #endif
                    atemDownstreamKeyerRight[keyer] = (int16_t) word(_pac
ketBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerRight[
keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemDownstreamKeyerRight[keyer=")
); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerRight[keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("DskS"))) {

                keyer = _packetBuffer[0];
                if (keyer<=1) {
                    #if ATEM_debug
                    temp = atemDownstreamKeyerOnAir[keyer];
                    #endif
                    atemDownstreamKeyerOnAir[keyer] = _packetBuffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerOnAir[
keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemDownstreamKeyerOnAir[keyer=")
); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerOnAir[keyer]);
                    }
                    #endif

                    #if ATEM_debug
```

```
                    temp = atemDownstreamKeyerInTransition[keyer];
                    #endif
                    atemDownstreamKeyerInTransition[keyer] = _packetBuffe
r[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerInTran
sition[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemDownstreamKeyerInTransition[k
eyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerInTransition[ke
yer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerIsAutoTransitioning[keyer];
                    #endif
                    atemDownstreamKeyerIsAutoTransitioning[keyer] = _pack
etBuffer[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerIsAuto
Transitioning[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())
) {
                        Serial.print(F("atemDownstreamKeyerIsAutoTransiti
oning[keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerIsAutoTransitio
ning[keyer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemDownstreamKeyerFramesRemaining[keyer];
                    #endif
                    atemDownstreamKeyerFramesRemaining[keyer] = _packetBu
ffer[4];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemDownstreamKeyerFrames
Remaining[keyer]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemDownstreamKeyerFramesRemainin
g[keyer=")); Serial.print(keyer); Serial.print(F("] = "));
                        Serial.println(atemDownstreamKeyerFramesRemaining
[keyer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("FtbP"))) {
```

```
                    mE = _packetBuffer[0];
                    if (mE<=1) {
                        #if ATEM_debug
                        temp = atemFadeToBlackRate[mE];
                        #endif
                        atemFadeToBlackRate[mE] = _packetBuffer[1];
                        #if ATEM_debug
                        if ((_serialOutput==0x80 && atemFadeToBlackRate[mE]!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                            Serial.print(F("atemFadeToBlackRate[mE=")); Seria
l.print(mE); Serial.print(F("] = "));
                            Serial.println(atemFadeToBlackRate[mE]);
                        }
                        #endif

                    }
                } else
                if(!strcmp(cmdStr, ("FtbS"))) {

                    mE = _packetBuffer[0];
                    if (mE<=1) {
                        #if ATEM_debug
                        temp = atemFadeToBlackStateFullyBlack[mE];
                        #endif
                        atemFadeToBlackStateFullyBlack[mE] = _packetBuffer[1]
;
                        #if ATEM_debug
                        if ((_serialOutput==0x80 && atemFadeToBlackStateFully
Black[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                            Serial.print(F("atemFadeToBlackStateFullyBlack[mE
=")); Serial.print(mE); Serial.print(F("] = "));
                            Serial.println(atemFadeToBlackStateFullyBlack[mE]
);
                        }
                        #endif

                        #if ATEM_debug
                        temp = atemFadeToBlackStateInTransition[mE];
                        #endif
                        atemFadeToBlackStateInTransition[mE] = _packetBuffer[
2];
                        #if ATEM_debug
                        if ((_serialOutput==0x80 && atemFadeToBlackStateInTra
nsition[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                            Serial.print(F("atemFadeToBlackStateInTransition[
mE=")); Serial.print(mE); Serial.print(F("] = "));
                            Serial.println(atemFadeToBlackStateInTransition[m
E]);
                        }
```

```
                    #endif

                    #if ATEM_debug
                    temp = atemFadeToBlackStateFramesRemaining[mE];
                    #endif
                    atemFadeToBlackStateFramesRemaining[mE] = _packetBuff
er[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemFadeToBlackStateFrame
sRemaining[mE]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemFadeToBlackStateFramesRemaini
ng[mE=")); Serial.print(mE); Serial.print(F("] = "));
                        Serial.println(atemFadeToBlackStateFramesRemainin
g[mE]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("ColV"))) {

                colorGenerator = _packetBuffer[0];
                if (colorGenerator<=1) {
                    #if ATEM_debug
                    temp = atemColorGeneratorHue[colorGenerator];
                    #endif
                    atemColorGeneratorHue[colorGenerator] = word(_packetB
uffer[2], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemColorGeneratorHue[col
orGenerator]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemColorGeneratorHue[colorGenera
tor=")); Serial.print(colorGenerator); Serial.print(F("] = "));
                        Serial.println(atemColorGeneratorHue[colorGenerat
or]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemColorGeneratorSaturation[colorGenerator];
                    #endif
                    atemColorGeneratorSaturation[colorGenerator] = word(_
packetBuffer[4], _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemColorGeneratorSaturat
ion[colorGenerator]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemColorGeneratorSaturation[colo
rGenerator=")); Serial.print(colorGenerator); Serial.print(F("] = "));
```

```
                        Serial.println(atemColorGeneratorSaturation[color
Generator]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemColorGeneratorLuma[colorGenerator];
                    #endif
                    atemColorGeneratorLuma[colorGenerator] = word(_packet
Buffer[6], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemColorGeneratorLuma[co
lorGenerator]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemColorGeneratorLuma[colorGener
ator=")); Serial.print(colorGenerator); Serial.print(F("] = "));
                        Serial.println(atemColorGeneratorLuma[colorGenera
tor]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("AuxS"))) {

                aUXChannel = _packetBuffer[0];
                if (aUXChannel<=5) {
                    #if ATEM_debug
                    temp = atemAuxSourceInput[aUXChannel];
                    #endif
                    atemAuxSourceInput[aUXChannel] = word(_packetBuffer[2
], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAuxSourceInput[aUXCha
nnel]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemAuxSourceInput[aUXChannel="))
; Serial.print(aUXChannel); Serial.print(F("] = "));
                        Serial.println(atemAuxSourceInput[aUXChannel]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("CCdP"))) {

                input = _packetBuffer[0];
                if (input<=20) {
                if (_packetBuffer[1]==0 && _packetBuffer[2]==3) {

                    #if ATEM_debug
```

```
                    temp = atemCameraControlIris[input];
                    #endif
                    atemCameraControlIris[input] = (int16_t) word(_packet
Buffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlIris[inp
ut]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemCameraControlIris[input="));
Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlIris[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==0 && _packetBuffer[2]==0) {

                    #if ATEM_debug
                    temp = atemCameraControlFocus[input];
                    #endif
                    atemCameraControlFocus[input] = (int16_t) word(_packe
tBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlFocus[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlFocus[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlFocus[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==1 && _packetBuffer[2]==1) {

                    #if ATEM_debug
                    temp = atemCameraControlGain[input];
                    #endif
                    atemCameraControlGain[input] = (int16_t) word(_packet
Buffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGain[inp
ut]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemCameraControlGain[input="));
Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGain[input]);
                    }
                    #endif
```

```
                }

                if (_packetBuffer[1]==1 && _packetBuffer[2]==2) {

                    #if ATEM_debug
                    temp = atemCameraControlWhiteBalance[input];
                    #endif
                    atemCameraControlWhiteBalance[input] = (int16_t) word
(_packetBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlWhiteBal
ance[input]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemCameraControlWhiteBalance[inp
ut=")); Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlWhiteBalance[inpu
t]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==1 && _packetBuffer[2]==8) {

                    #if ATEM_debug
                    temp = atemCameraControlSharpeningLevel[input];
                    #endif
                    atemCameraControlSharpeningLevel[input] = (int16_t) w
ord(_packetBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlSharpeni
ngLevel[input]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemCameraControlSharpeningLevel[
input=")); Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlSharpeningLevel[i
nput]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==0 && _packetBuffer[2]==8) {

                    #if ATEM_debug
                    temp = atemCameraControlZoomNormalized[input];
                    #endif
                    atemCameraControlZoomNormalized[input] = (int16_t) wo
rd(_packetBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemCameraControlZoomNorm
alized[input]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemCameraControlZoomNormalized[i
nput=")); Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlZoomNormalized[in
put]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==0 && _packetBuffer[2]==9) {

                    #if ATEM_debug
                    temp = atemCameraControlZoomSpeed[input];
                    #endif
                    atemCameraControlZoomSpeed[input] = (int16_t) word(_p
acketBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlZoomSpee
d[input]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlZoomSpeed[input=
")); Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlZoomSpeed[input])
;
                    }
                    #endif

                }

                if (_packetBuffer[1]==4 && _packetBuffer[2]==4) {

                    #if ATEM_debug
                    temp = atemCameraControlColorbars[input];
                    #endif
                    atemCameraControlColorbars[input] = (int16_t) word(_p
acketBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlColorbar
s[input]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlColorbars[input=
")); Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlColorbars[input])
;
                    }
                    #endif

                }
```

```
                if (_packetBuffer[1]==8 && _packetBuffer[2]==0) {

                    #if ATEM_debug
                    temp = atemCameraControlLiftR[input];
                    #endif
                    atemCameraControlLiftR[input] = (int16_t) word(_packe
tBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlLiftR[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlLiftR[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlLiftR[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==1) {

                    #if ATEM_debug
                    temp = atemCameraControlGammaR[input];
                    #endif
                    atemCameraControlGammaR[input] = (int16_t) word(_pack
etBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGammaR[i
nput]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemCameraControlGammaR[input="))
; Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGammaR[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==2) {

                    #if ATEM_debug
                    temp = atemCameraControlGainR[input];
                    #endif
                    atemCameraControlGainR[input] = (int16_t) word(_packe
tBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGainR[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlGainR[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGainR[input]);
```

```
                }
                #endif

            }

            if (_packetBuffer[1]==8 && _packetBuffer[2]==5) {

                #if ATEM_debug
                temp = atemCameraControlLumMix[input];
                #endif
                atemCameraControlLumMix[input] = (int16_t) word(_pack
etBuffer[16], _packetBuffer[17]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemCameraControlLumMix[i
nput]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                    Serial.print(F("atemCameraControlLumMix[input="))
; Serial.print(input); Serial.print(F("] = "));
                    Serial.println(atemCameraControlLumMix[input]);
                }
                #endif

            }

            if (_packetBuffer[1]==8 && _packetBuffer[2]==6) {

                #if ATEM_debug
                temp = atemCameraControlHue[input];
                #endif
                atemCameraControlHue[input] = (int16_t) word(_packetB
uffer[16], _packetBuffer[17]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemCameraControlHue[inpu
t]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                    Serial.print(F("atemCameraControlHue[input=")); S
erial.print(input); Serial.print(F("] = "));
                    Serial.println(atemCameraControlHue[input]);
                }
                #endif

            }

            if (_packetBuffer[1]==1 && _packetBuffer[2]==5) {

                #if ATEM_debug
                temp = atemCameraControlShutter[input];
                #endif
                atemCameraControlShutter[input] = (int16_t) word(_pac
ketBuffer[18], _packetBuffer[19]);
                #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemCameraControlShutter[
input]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemCameraControlShutter[input=")
); Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlShutter[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==0) {

                    #if ATEM_debug
                    temp = atemCameraControlLiftG[input];
                    #endif
                    atemCameraControlLiftG[input] = (int16_t) word(_packe
tBuffer[18], _packetBuffer[19]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlLiftG[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlLiftG[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlLiftG[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==1) {

                    #if ATEM_debug
                    temp = atemCameraControlGammaG[input];
                    #endif
                    atemCameraControlGammaG[input] = (int16_t) word(_pack
etBuffer[18], _packetBuffer[19]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGammaG[i
nput]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemCameraControlGammaG[input="))
; Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGammaG[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==2) {

                    #if ATEM_debug
```

```
                    temp = atemCameraControlGainG[input];
                    #endif
                    atemCameraControlGainG[input] = (int16_t) word(_packe
tBuffer[18], _packetBuffer[19]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGainG[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlGainG[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGainG[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==4) {

                    #if ATEM_debug
                    temp = atemCameraControlContrast[input];
                    #endif
                    atemCameraControlContrast[input] = (int16_t) word(_pa
cketBuffer[18], _packetBuffer[19]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlContrast
[input]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemCameraControlContrast[input="
)); Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlContrast[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==6) {

                    #if ATEM_debug
                    temp = atemCameraControlSaturation[input];
                    #endif
                    atemCameraControlSaturation[input] = (int16_t) word(_
packetBuffer[18], _packetBuffer[19]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlSaturati
on[input]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemCameraControlSaturation[input
=")); Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlSaturation[input]
);
                    }
                    #endif
```

```
                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==0) {

                    #if ATEM_debug
                    temp = atemCameraControlLiftB[input];
                    #endif
                    atemCameraControlLiftB[input] = (int16_t) word(_packe
tBuffer[20], _packetBuffer[21]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlLiftB[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlLiftB[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlLiftB[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==1) {

                    #if ATEM_debug
                    temp = atemCameraControlGammaB[input];
                    #endif
                    atemCameraControlGammaB[input] = (int16_t) word(_pack
etBuffer[20], _packetBuffer[21]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGammaB[i
nput]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemCameraControlGammaB[input="))
; Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGammaB[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==2) {

                    #if ATEM_debug
                    temp = atemCameraControlGainB[input];
                    #endif
                    atemCameraControlGainB[input] = (int16_t) word(_packe
tBuffer[20], _packetBuffer[21]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGainB[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
```

```
                        Serial.print(F("atemCameraControlGainB[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGainB[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==0) {

                    #if ATEM_debug
                    temp = atemCameraControlLiftY[input];
                    #endif
                    atemCameraControlLiftY[input] = (int16_t) word(_packe
tBuffer[22], _packetBuffer[23]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlLiftY[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlLiftY[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlLiftY[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==1) {

                    #if ATEM_debug
                    temp = atemCameraControlGammaY[input];
                    #endif
                    atemCameraControlGammaY[input] = (int16_t) word(_pack
etBuffer[22], _packetBuffer[23]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGammaY[i
nput]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemCameraControlGammaY[input="))
; Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGammaY[input]);
                    }
                    #endif

                }

                if (_packetBuffer[1]==8 && _packetBuffer[2]==2) {

                    #if ATEM_debug
                    temp = atemCameraControlGainY[input];
                    #endif
```

```
                    atemCameraControlGainY[input] = (int16_t) word(_packe
tBuffer[22], _packetBuffer[23]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemCameraControlGainY[in
put]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemCameraControlGainY[input="));
 Serial.print(input); Serial.print(F("] = "));
                        Serial.println(atemCameraControlGainY[input]);
                    }
                    #endif

                }

                }
            } else
            if(!strcmp(cmdStr, ("RCPS"))) {

                mediaPlayer = _packetBuffer[0];
                if (mediaPlayer<=1) {
                    #if ATEM_debug
                    temp = atemClipPlayerPlaying[mediaPlayer];
                    #endif
                    atemClipPlayerPlaying[mediaPlayer] = _packetBuffer[1]
;
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemClipPlayerPlaying[med
iaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemClipPlayerPlaying[mediaPlayer
=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemClipPlayerPlaying[mediaPlayer]
);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemClipPlayerLoop[mediaPlayer];
                    #endif
                    atemClipPlayerLoop[mediaPlayer] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemClipPlayerLoop[mediaP
layer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemClipPlayerLoop[mediaPlayer=")
); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemClipPlayerLoop[mediaPlayer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemClipPlayerAtBeginning[mediaPlayer];
```

```
                    #endif
                    atemClipPlayerAtBeginning[mediaPlayer] = _packetBuffe
r[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemClipPlayerAtBeginning
[mediaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemClipPlayerAtBeginning[mediaPl
ayer=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemClipPlayerAtBeginning[mediaPla
yer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemClipPlayerClipFrame[mediaPlayer];
                    #endif
                    atemClipPlayerClipFrame[mediaPlayer] = word(_packetBu
ffer[4], _packetBuffer[5]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemClipPlayerClipFrame[m
ediaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemClipPlayerClipFrame[mediaPlay
er=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemClipPlayerClipFrame[mediaPlaye
r]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("MPCE"))) {

                mediaPlayer = _packetBuffer[0];
                if (mediaPlayer<=1) {
                    #if ATEM_debug
                    temp = atemMediaPlayerSourceType[mediaPlayer];
                    #endif
                    atemMediaPlayerSourceType[mediaPlayer] = _packetBuffe
r[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerSourceType
[mediaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMediaPlayerSourceType[mediaPl
ayer=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerSourceType[mediaPla
yer]);
                    }
                    #endif
```

```
                    #if ATEM_debug
                    temp = atemMediaPlayerSourceStillIndex[mediaPlayer];
                    #endif
                    atemMediaPlayerSourceStillIndex[mediaPlayer] = _packe
tBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerSourceStil
lIndex[mediaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemMediaPlayerSourceStillIndex[m
ediaPlayer=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerSourceStillIndex[me
diaPlayer]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemMediaPlayerSourceClipIndex[mediaPlayer];
                    #endif
                    atemMediaPlayerSourceClipIndex[mediaPlayer] = _packet
Buffer[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerSourceClip
Index[mediaPlayer]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemMediaPlayerSourceClipIndex[me
diaPlayer=")); Serial.print(mediaPlayer); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerSourceClipIndex[med
iaPlayer]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("MPSp"))) {

                    #if ATEM_debug
                    temp = atemMediaPoolStorageClip1MaxLength;
                    #endif
                    atemMediaPoolStorageClip1MaxLength = word(_packetBuff
er[0], _packetBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPoolStorageClip1
MaxLength!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMediaPoolStorageClip1MaxLengt
h = "));

                        Serial.println(atemMediaPoolStorageClip1MaxLength
);
                    }
```

```
                    #endif

                    #if ATEM_debug
                    temp = atemMediaPoolStorageClip2MaxLength;
                    #endif
                    atemMediaPoolStorageClip2MaxLength = word(_packetBuff
er[2], _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPoolStorageClip2
MaxLength!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMediaPoolStorageClip2MaxLengt
h = "));

                        Serial.println(atemMediaPoolStorageClip2MaxLength
);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("MPCS"))) {

                clipBank = _packetBuffer[0];
                if (clipBank<=1) {
                    #if ATEM_debug
                    temp = atemMediaPlayerClipSourceIsUsed[clipBank];
                    #endif
                    atemMediaPlayerClipSourceIsUsed[clipBank] = _packetBu
ffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerClipSource
IsUsed[clipBank]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemMediaPlayerClipSourceIsUsed[c
lipBank=")); Serial.print(clipBank); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerClipSourceIsUsed[cl
ipBank]);
                    }
                    #endif

                    memset(atemMediaPlayerClipSourceFileName[clipBank],0,
17);
                    strncpy(atemMediaPlayerClipSourceFileName[clipBank],
(char *)(_packetBuffer+2), 16);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && hasInitialized()) || (_se
rialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMediaPlayerClipSourceFileName
[clipBank=")); Serial.print(clipBank); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerClipSourceFileName[
clipBank]);
                    }
```

```
                    #endif

                    #if ATEM_debug
                    temp = atemMediaPlayerClipSourceFrames[clipBank];
                    #endif
                    atemMediaPlayerClipSourceFrames[clipBank] = word(_pac
ketBuffer[66], _packetBuffer[67]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerClipSource
Frames[clipBank]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemMediaPlayerClipSourceFrames[c
lipBank=")); Serial.print(clipBank); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerClipSourceFrames[cl
ipBank]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("MPAS"))) {

                clipBank = _packetBuffer[0];
                if (clipBank<=2) {
                    #if ATEM_debug
                    temp = atemMediaPlayerAudioSourceIsUsed[clipBank];
                    #endif
                    atemMediaPlayerAudioSourceIsUsed[clipBank] = _packetB
uffer[1];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMediaPlayerAudioSourc
eIsUsed[clipBank]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemMediaPlayerAudioSourceIsUsed[
clipBank=")); Serial.print(clipBank); Serial.print(F("] = "));
                        Serial.println(atemMediaPlayerAudioSourceIsUsed[c
lipBank]);
                    }
                    #endif

                    memset(atemMediaPlayerAudioSourceFileName[clipBank],0
,17);
                    strncpy(atemMediaPlayerAudioSourceFileName[clipBank],
 (char *)(_packetBuffer+18), 16);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && hasInitialized()) || (_se
rialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemMediaPlayerAudioSourceFileNam
e[clipBank=")); Serial.print(clipBank); Serial.print(F("] = "));
```

```
                    Serial.println(atemMediaPlayerAudioSourceFileName
[clipBank]);
                }
                #endif

            }
        } else
        if(!strcmp(cmdStr, ("MRPr"))) {

                #if ATEM_debug
                temp = atemMacroRunStatusState;
                #endif
                atemMacroRunStatusState = _packetBuffer[0];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemMacroRunStatusState!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                    Serial.print(F("atemMacroRunStatusState = "));
                    Serial.println(atemMacroRunStatusState);
                }
                #endif

                #if ATEM_debug
                temp = atemMacroRunStatusIsLooping;
                #endif
                atemMacroRunStatusIsLooping = _packetBuffer[1];
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemMacroRunStatusIsLoopi
ng!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                    Serial.print(F("atemMacroRunStatusIsLooping = "))
;
                    Serial.println(atemMacroRunStatusIsLooping);
                }
                #endif

                #if ATEM_debug
                temp = atemMacroRunStatusIndex;
                #endif
                atemMacroRunStatusIndex = word(_packetBuffer[2], _pac
ketBuffer[3]);
                #if ATEM_debug
                if ((_serialOutput==0x80 && atemMacroRunStatusIndex!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                    Serial.print(F("atemMacroRunStatusIndex = "));
                    Serial.println(atemMacroRunStatusIndex);
                }
                #endif

        } else
        if(!strcmp(cmdStr, ("MPrp"))) {
```

```
                        macroIndex = _packetBuffer[1];
                        if (macroIndex<=9) {
                            #if ATEM_debug
                            temp = atemMacroPropertiesIsUsed[macroIndex];
                            #endif
                            atemMacroPropertiesIsUsed[macroIndex] = _packetBuffer
[2];
                            #if ATEM_debug
                            if ((_serialOutput==0x80 && atemMacroPropertiesIsUsed
[macroIndex]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                                Serial.print(F("atemMacroPropertiesIsUsed[macroIn
dex=")); Serial.print(macroIndex); Serial.print(F("] = "));
                                Serial.println(atemMacroPropertiesIsUsed[macroInd
ex]);
                            }
                            #endif

                            memset(atemMacroPropertiesName[macroIndex],0,11);
                            strncpy(atemMacroPropertiesName[macroIndex], (char *)
(_packetBuffer+8), _packetBuffer[5] > 10 ? 10 : _packetBuffer[5]);
                            #if ATEM_debug
                            if ((_serialOutput==0x80 && hasInitialized()) || (_se
rialOutput==0x81 && !hasInitialized()))    {
                                Serial.print(F("atemMacroPropertiesName[macroInde
x=")); Serial.print(macroIndex); Serial.print(F("] = "));
                                Serial.println(atemMacroPropertiesName[macroIndex
]);
                            }
                            #endif

                        }
                    } else
                    if(!strcmp(cmdStr, ("MRcS"))) {

                            #if ATEM_debug
                            temp = atemMacroRecordingStatusIsRecording;
                            #endif
                            atemMacroRecordingStatusIsRecording = _packetBuffer[0
];
                            #if ATEM_debug
                            if ((_serialOutput==0x80 && atemMacroRecordingStatusI
sRecording!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                                Serial.print(F("atemMacroRecordingStatusIsRecordi
ng = "));

                                Serial.println(atemMacroRecordingStatusIsRecordin
g);
                            }
                            #endif
```

```
                    #if ATEM_debug
                    temp = atemMacroRecordingStatusIndex;
                    #endif
                    atemMacroRecordingStatusIndex = word(_packetBuffer[2]
, _packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemMacroRecordingStatusI
ndex!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemMacroRecordingStatusIndex = "
));
                        Serial.println(atemMacroRecordingStatusIndex);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("SSrc"))) {

                    #if ATEM_debug
                    temp = atemSuperSourceFillSource;
                    #endif
                    atemSuperSourceFillSource = word(_packetBuffer[0], _p
acketBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceFillSource
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemSuperSourceFillSource = "));
                        Serial.println(atemSuperSourceFillSource);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceKeySource;
                    #endif
                    atemSuperSourceKeySource = word(_packetBuffer[2], _pa
cketBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceKeySource!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemSuperSourceKeySource = "));
                        Serial.println(atemSuperSourceKeySource);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceForeground;
                    #endif
                    atemSuperSourceForeground = _packetBuffer[4];
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemSuperSourceForeground
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemSuperSourceForeground = "));
                        Serial.println(atemSuperSourceForeground);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourcePreMultiplied;
                    #endif
                    atemSuperSourcePreMultiplied = _packetBuffer[5];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourcePreMultipl
ied!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemSuperSourcePreMultiplied = ")
);
                        Serial.println(atemSuperSourcePreMultiplied);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceClip;
                    #endif
                    atemSuperSourceClip = word(_packetBuffer[6], _packetB
uffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceClip!=temp
) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemSuperSourceClip = "));
                        Serial.println(atemSuperSourceClip);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceGain;
                    #endif
                    atemSuperSourceGain = word(_packetBuffer[8], _packetB
uffer[9]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceGain!=temp
) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemSuperSourceGain = "));
                        Serial.println(atemSuperSourceGain);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceInvertKey;
                    #endif
```

```
                    atemSuperSourceInvertKey = _packetBuffer[10];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceInvertKey!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemSuperSourceInvertKey = "));
                        Serial.println(atemSuperSourceInvertKey);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderEnabled;
                    #endif
                    atemSuperSourceBorderEnabled = _packetBuffer[11];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderEnab
led!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemSuperSourceBorderEnabled = ")
);

                        Serial.println(atemSuperSourceBorderEnabled);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderBevel;
                    #endif
                    atemSuperSourceBorderBevel = _packetBuffer[12];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderBeve
l!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemSuperSourceBorderBevel = "));
                        Serial.println(atemSuperSourceBorderBevel);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderOuterWidth;
                    #endif
                    atemSuperSourceBorderOuterWidth = word(_packetBuffer[
14], _packetBuffer[15]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderOute
rWidth!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemSuperSourceBorderOuterWidth =
 "));

                        Serial.println(atemSuperSourceBorderOuterWidth);
                    }
                    #endif

                    #if ATEM_debug
```

```
                    temp = atemSuperSourceBorderInnerWidth;
                    #endif
                    atemSuperSourceBorderInnerWidth = word(_packetBuffer[
16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderInne
rWidth!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemSuperSourceBorderInnerWidth =
 "));

                        Serial.println(atemSuperSourceBorderInnerWidth);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderOuterSoftness;
                    #endif
                    atemSuperSourceBorderOuterSoftness = _packetBuffer[18
];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderOute
rSoftness!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemSuperSourceBorderOuterSoftnes
s = "));

                        Serial.println(atemSuperSourceBorderOuterSoftness
);

                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderInnerSoftness;
                    #endif
                    atemSuperSourceBorderInnerSoftness = _packetBuffer[19
];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderInne
rSoftness!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemSuperSourceBorderInnerSoftnes
s = "));

                        Serial.println(atemSuperSourceBorderInnerSoftness
);

                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderBevelSoftness;
                    #endif
                    atemSuperSourceBorderBevelSoftness = _packetBuffer[20
];
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemSuperSourceBorderBeve
lSoftness!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemSuperSourceBorderBevelSoftnes
s = "));
                        Serial.println(atemSuperSourceBorderBevelSoftness
);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderBevelPosition;
                    #endif
                    atemSuperSourceBorderBevelPosition = _packetBuffer[21
];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderBeve
lPosition!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemSuperSourceBorderBevelPositio
n = "));
                        Serial.println(atemSuperSourceBorderBevelPosition
);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderHue;
                    #endif
                    atemSuperSourceBorderHue = word(_packetBuffer[22], _p
acketBuffer[23]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderHue!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemSuperSourceBorderHue = "));
                        Serial.println(atemSuperSourceBorderHue);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderSaturation;
                    #endif
                    atemSuperSourceBorderSaturation = word(_packetBuffer[
24], _packetBuffer[25]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderSatu
ration!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemSuperSourceBorderSaturation =
 "));
                        Serial.println(atemSuperSourceBorderSaturation);
                    }
```

```
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBorderLuma;
                    #endif
                    atemSuperSourceBorderLuma = word(_packetBuffer[26], _
packetBuffer[27]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBorderLuma
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemSuperSourceBorderLuma = "));
                        Serial.println(atemSuperSourceBorderLuma);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceLightSourceDirection;
                    #endif
                    atemSuperSourceLightSourceDirection = word(_packetBuf
fer[28], _packetBuffer[29]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceLightSourc
eDirection!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemSuperSourceLightSourceDirecti
on = "));

                        Serial.println(atemSuperSourceLightSourceDirectio
n);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceLightSourceAltitude;
                    #endif
                    atemSuperSourceLightSourceAltitude = _packetBuffer[30
];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceLightSourc
eAltitude!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemSuperSourceLightSourceAltitud
e = "));

                        Serial.println(atemSuperSourceLightSourceAltitude
);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("SSBP"))) {

                box = _packetBuffer[0];
```

```
                  if (box<=3) {
                      #if ATEM_debug
                      temp = atemSuperSourceBoxParametersEnabled[box];
                      #endif
                      atemSuperSourceBoxParametersEnabled[box] = _packetBuf
fer[1];
                      #if ATEM_debug
                      if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersEnabled[box]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                          Serial.print(F("atemSuperSourceBoxParametersEnabl
ed[box=")); Serial.print(box); Serial.print(F("] = "));
                          Serial.println(atemSuperSourceBoxParametersEnable
d[box]);
                      }
                      #endif

                      #if ATEM_debug
                      temp = atemSuperSourceBoxParametersInputSource[box];
                      #endif
                      atemSuperSourceBoxParametersInputSource[box] = word(_
packetBuffer[2], _packetBuffer[3]);
                      #if ATEM_debug
                      if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersInputSource[box]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                          Serial.print(F("atemSuperSourceBoxParametersInput
Source[box=")); Serial.print(box); Serial.print(F("] = "));
                          Serial.println(atemSuperSourceBoxParametersInputS
ource[box]);
                      }
                      #endif

                      #if ATEM_debug
                      temp = atemSuperSourceBoxParametersPositionX[box];
                      #endif
                      atemSuperSourceBoxParametersPositionX[box] = (int16_t
) word(_packetBuffer[4], _packetBuffer[5]);
                      #if ATEM_debug
                      if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersPositionX[box]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                          Serial.print(F("atemSuperSourceBoxParametersPosit
ionX[box=")); Serial.print(box); Serial.print(F("] = "));
                          Serial.println(atemSuperSourceBoxParametersPositi
onX[box]);
                      }
                      #endif

                      #if ATEM_debug
```

```
                    temp = atemSuperSourceBoxParametersPositionY[box];
                    #endif
                    atemSuperSourceBoxParametersPositionY[box] = (int16_t
) word(_packetBuffer[6], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersPositionY[box]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemSuperSourceBoxParametersPosit
ionY[box=")); Serial.print(box); Serial.print(F("] = "));
                        Serial.println(atemSuperSourceBoxParametersPositi
onY[box]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBoxParametersSize[box];
                    #endif
                    atemSuperSourceBoxParametersSize[box] = word(_packetB
uffer[8], _packetBuffer[9]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersSize[box]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemSuperSourceBoxParametersSize[
box=")); Serial.print(box); Serial.print(F("] = "));
                        Serial.println(atemSuperSourceBoxParametersSize[b
ox]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBoxParametersCropped[box];
                    #endif
                    atemSuperSourceBoxParametersCropped[box] = _packetBuf
fer[10];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersCropped[box]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemSuperSourceBoxParametersCropp
ed[box=")); Serial.print(box); Serial.print(F("] = "));
                        Serial.println(atemSuperSourceBoxParametersCroppe
d[box]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBoxParametersCropTop[box];
                    #endif
```

```
                    atemSuperSourceBoxParametersCropTop[box] = word(_pack
etBuffer[12], _packetBuffer[13]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersCropTop[box]!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemSuperSourceBoxParametersCropT
op[box=")); Serial.print(box); Serial.print(F("] = "));
                        Serial.println(atemSuperSourceBoxParametersCropTo
p[box]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBoxParametersCropBottom[box];
                    #endif
                    atemSuperSourceBoxParametersCropBottom[box] = word(_p
acketBuffer[14], _packetBuffer[15]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersCropBottom[box]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemSuperSourceBoxParametersCropB
ottom[box=")); Serial.print(box); Serial.print(F("] = "));
                        Serial.println(atemSuperSourceBoxParametersCropBo
ttom[box]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBoxParametersCropLeft[box];
                    #endif
                    atemSuperSourceBoxParametersCropLeft[box] = word(_pac
ketBuffer[16], _packetBuffer[17]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersCropLeft[box]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemSuperSourceBoxParametersCropL
eft[box=")); Serial.print(box); Serial.print(F("] = "));
                        Serial.println(atemSuperSourceBoxParametersCropLe
ft[box]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemSuperSourceBoxParametersCropRight[box];
                    #endif
                    atemSuperSourceBoxParametersCropRight[box] = word(_pa
cketBuffer[18], _packetBuffer[19]);
                    #if ATEM_debug
```

```
                    if ((_serialOutput==0x80 && atemSuperSourceBoxParamet
ersCropRight[box]!=temp) || (_serialOutput==0x81 && !hasInitialized()))
  {
                        Serial.print(F("atemSuperSourceBoxParametersCropR
ight[box=")); Serial.print(box); Serial.print(F("] = "));
                        Serial.println(atemSuperSourceBoxParametersCropRi
ght[box]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("AMIP"))) {

                audioSource = word(_packetBuffer[0],_packetBuffer[1]);
                if (getAudioSrcIndex(audioSource)<=24) {
                    #if ATEM_debug
                    temp = atemAudioMixerInputType[getAudioSrcIndex(audio
Source)];
                    #endif
                    atemAudioMixerInputType[getAudioSrcIndex(audioSource)
] = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputType[g
etAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 && !hasIniti
alized()))    {
                        Serial.print(F("atemAudioMixerInputType[getAudioS
rcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioSource)); Se
rial.print(F("] = "));
                        Serial.println(atemAudioMixerInputType[getAudioSr
cIndex(audioSource)]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerInputFromMediaPlayer[getAudioSrc
Index(audioSource)];
                    #endif
                    atemAudioMixerInputFromMediaPlayer[getAudioSrcIndex(a
udioSource)] = _packetBuffer[6];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputFromMe
diaPlayer[getAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 &
& !hasInitialized())) {
                        Serial.print(F("atemAudioMixerInputFromMediaPlaye
r[getAudioSrcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioS
ource)); Serial.print(F("] = "));
                        Serial.println(atemAudioMixerInputFromMediaPlayer
[getAudioSrcIndex(audioSource)]);
```

```
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerInputPlugtype[getAudioSrcIndex(a
udioSource)];
                    #endif
                    atemAudioMixerInputPlugtype[getAudioSrcIndex(audioSou
rce)] = _packetBuffer[7];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputPlugty
pe[getAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 && !hasI
nitialized()))    {
                        Serial.print(F("atemAudioMixerInputPlugtype[getAu
dioSrcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioSource))
; Serial.print(F("] = "));
                        Serial.println(atemAudioMixerInputPlugtype[getAud
ioSrcIndex(audioSource)]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerInputMixOption[getAudioSrcIndex(
audioSource)];
                    #endif
                    atemAudioMixerInputMixOption[getAudioSrcIndex(audioSo
urce)] = _packetBuffer[8];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputMixOpt
ion[getAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 && !has
Initialized()))   {
                        Serial.print(F("atemAudioMixerInputMixOption[getA
udioSrcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioSource)
); Serial.print(F("] = "));
                        Serial.println(atemAudioMixerInputMixOption[getAu
dioSrcIndex(audioSource)]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerInputVolume[getAudioSrcIndex(aud
ioSource)];
                    #endif
                    atemAudioMixerInputVolume[getAudioSrcIndex(audioSourc
e)] = word(_packetBuffer[10], _packetBuffer[11]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputVolume
[getAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 && !hasIni
tialized()))  {
```

```
                        Serial.print(F("atemAudioMixerInputVolume[getAudi
oSrcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioSource));
Serial.print(F("] = "));
                        Serial.println(atemAudioMixerInputVolume[getAudio
SrcIndex(audioSource)]);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerInputBalance[getAudioSrcIndex(au
dioSource)];
                    #endif
                    atemAudioMixerInputBalance[getAudioSrcIndex(audioSour
ce)] = (int16_t) word(_packetBuffer[12], _packetBuffer[13]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerInputBalanc
e[getAudioSrcIndex(audioSource)]!=temp) || (_serialOutput==0x81 && !hasIn
itialized())) {
                        Serial.print(F("atemAudioMixerInputBalance[getAud
ioSrcIndex(audioSource)=")); Serial.print(getAudioSrcIndex(audioSource));
 Serial.print(F("] = "));
                        Serial.println(atemAudioMixerInputBalance[getAudi
oSrcIndex(audioSource)]);
                    }
                    #endif

                }
            } else
            if(!strcmp(cmdStr, ("AMMO"))) {

                    #if ATEM_debug
                    temp = atemAudioMixerMasterVolume;
                    #endif
                    atemAudioMixerMasterVolume = word(_packetBuffer[0], _
packetBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerMasterVolum
e!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemAudioMixerMasterVolume = "));
                        Serial.println(atemAudioMixerMasterVolume);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("AMmO"))) {

                    #if ATEM_debug
                    temp = atemAudioMixerMonitorMonitorAudio;
                    #endif
```

```
                    atemAudioMixerMonitorMonitorAudio = _packetBuffer[0];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerMonitorMoni
torAudio!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemAudioMixerMonitorMonitorAudio
 = "));
                        Serial.println(atemAudioMixerMonitorMonitorAudio)
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerMonitorVolume;
                    #endif
                    atemAudioMixerMonitorVolume = word(_packetBuffer[2],
_packetBuffer[3]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerMonitorVolu
me!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemAudioMixerMonitorVolume = "))
;
                        Serial.println(atemAudioMixerMonitorVolume);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerMonitorMute;
                    #endif
                    atemAudioMixerMonitorMute = _packetBuffer[4];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerMonitorMute
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemAudioMixerMonitorMute = "));
                        Serial.println(atemAudioMixerMonitorMute);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerMonitorSolo;
                    #endif
                    atemAudioMixerMonitorSolo = _packetBuffer[5];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerMonitorSolo
!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemAudioMixerMonitorSolo = "));
                        Serial.println(atemAudioMixerMonitorSolo);
                    }
                    #endif
```

```
                    #if ATEM_debug
                    temp = atemAudioMixerMonitorSoloInput;
                    #endif
                    atemAudioMixerMonitorSoloInput = word(_packetBuffer[6
], _packetBuffer[7]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerMonitorSolo
Input!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemAudioMixerMonitorSoloInput =
"));

                        Serial.println(atemAudioMixerMonitorSoloInput);
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemAudioMixerMonitorDim;
                    #endif
                    atemAudioMixerMonitorDim = _packetBuffer[8];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerMonitorDim!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemAudioMixerMonitorDim = "));
                        Serial.println(atemAudioMixerMonitorDim);
                    }
                    #endif

            } else
            if(!strcmp(cmdStr, ("AMLv"))) {

                sources = word(_packetBuffer[0],_packetBuffer[1]);
                if (sources<=24) {

                        atemAudioMixerLevelsSources = word(_packetBuffer[
0], _packetBuffer[1]);
                            //Serial.print(F("atemAudioMixerLevelsSources
 = "));

                            //Serial.println(atemAudioMixerLevelsSources)
;

                        atemAudioMixerLevelsMasterLeft = (uint16_t)_packe
tBuffer[5]<<8 | _packetBuffer[6];
                            //Serial.print(F("atemAudioMixerLevelsMasterL
eft = "));

                            //Serial.println(atemAudioMixerLevelsMasterLe
ft);

                        atemAudioMixerLevelsMasterRight = (uint16_t)_pack
etBuffer[9]<<8 | _packetBuffer[10];
```

```
                            //Serial.print(F("atemAudioMixerLevelsMasterR
ight = "));
                            //Serial.println(atemAudioMixerLevelsMasterRi
ght);

                        atemAudioMixerLevelsMasterPeakLeft = (uint16_t)_p
acketBuffer[13]<<8 | _packetBuffer[14];
                            //Serial.print(F("atemAudioMixerLevelsMasterP
eakLeft = "));
                            //Serial.println(atemAudioMixerLevelsMasterPe
akLeft);

                        atemAudioMixerLevelsMasterPeakRight = (uint16_t)_
packetBuffer[17]<<8 | _packetBuffer[18];
                            //Serial.print(F("atemAudioMixerLevelsMasterP
eakRight = "));
                            //Serial.println(atemAudioMixerLevelsMasterPe
akRight);

                        atemAudioMixerLevelsMonitor = (uint16_t)_packetBu
ffer[21]<<8 | _packetBuffer[22];
                            //Serial.print(F("atemAudioMixerLevelsMonitor
 = "));
                            //Serial.println(atemAudioMixerLevelsMonitor)
;

                        _readToPacketBuffer(sources*2);
                        for(uint8_t a=0;a<sources;a++)  {
                            atemAudioMixerLevelsSourceOrder[a] = word(_pa
cketBuffer[a<<1], _packetBuffer[(a<<1)+1]);
                            //Serial.print(F("atemAudioMixerLevelsSourceO
rder[a=")); Serial.print(a); Serial.print(F("] = "));
                            //Serial.println(atemAudioMixerLevelsSourceOr
der[a]);
                        }
                        if (sources&0xB1)   {   // We must read 4-
byte chunks, so compensate if sources was an odd number
                            _readToPacketBuffer(2);
                        }

                        for(uint8_t a=0;a<sources;a++)  {
                            _readToPacketBuffer(16);

                            atemAudioMixerLevelsSourceLeft[a] = (uint16_t
)_packetBuffer[1]<<8 | _packetBuffer[2];
                            //  Serial.print(F("atemAudioMixerLevelsSourc
eLeft[a=")); Serial.print(a); Serial.print(F("] = "));
                            //  Serial.println(atemAudioMixerLevelsSource
Left[a]);
```

```
                            atemAudioMixerLevelsSourceRight[a] = (uint16_
t)_packetBuffer[5]<<8 | _packetBuffer[6];
                            //  Serial.print(F("atemAudioMixerLevelsSourc
eRight[a=")); Serial.print(a); Serial.print(F("] = "));
                            //  Serial.println(atemAudioMixerLevelsSource
Right[a]);

                            atemAudioMixerLevelsSourcePeakLeft[a] = (uint
16_t)_packetBuffer[9]<<8 | _packetBuffer[10];
                            //  Serial.print(F("atemAudioMixerLevelsSourc
ePeakLeft[a=")); Serial.print(a); Serial.print(F("] = "));
                            //  Serial.println(atemAudioMixerLevelsSource
PeakLeft[a]);

                            atemAudioMixerLevelsSourcePeakRight[a] = (uin
t16_t)_packetBuffer[13]<<8 | _packetBuffer[14];
                            //  Serial.print(F("atemAudioMixerLevelsSourc
ePeakRight[a=")); Serial.print(a); Serial.print(F("] = "));
                            //  Serial.println(atemAudioMixerLevelsSource
PeakRight[a]);
                        }

                }
            } else
            if(!strcmp(cmdStr, ("AMTl"))) {

                sources = word(_packetBuffer[0],_packetBuffer[1]);
                if (sources<=24) {

                    #if ATEM_debug
                    temp = atemAudioMixerTallySources;
                    #endif
                    atemAudioMixerTallySources = word(_packetBuffer[0], _
packetBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemAudioMixerTallySource
s!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                        Serial.print(F("atemAudioMixerTallySources = "));
                        Serial.println(atemAudioMixerTallySources);
                    }
                    #endif

                    for(uint8_t a=0;a<sources;a++)  {
                        #if ATEM_debug
                        temp = atemAudioMixerTallyAudioSource[a];
                        #endif
                        atemAudioMixerTallyAudioSource[a] = word(_packetB
uffer[2+3*a], _packetBuffer[2+(3*a)+1]);
```

```
                        #if ATEM_debug
                        if ((_serialOutput==0x80 && atemAudioMixerTallyAu
dioSource[a]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                            Serial.print(F("atemAudioMixerTallyAudioSourc
e[a=")); Serial.print(a); Serial.print(F("] = "));
                            Serial.println(atemAudioMixerTallyAudioSource
[a]);
                        }
                        #endif

                        #if ATEM_debug
                        temp = atemAudioMixerTallyIsMixedIn[a];
                        #endif
                        atemAudioMixerTallyIsMixedIn[a] = _packetBuffer[2
+(3*a)+2];
                        #if ATEM_debug
                        if ((_serialOutput==0x80 && atemAudioMixerTallyIs
MixedIn[a]!=temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                            Serial.print(F("atemAudioMixerTallyIsMixedIn[
a=")); Serial.print(a); Serial.print(F("] = "));
                            Serial.println(atemAudioMixerTallyIsMixedIn[a
]);
                        }
                        #endif
                    }

                }
            } else
            if(!strcmp(cmdStr, ("TlIn"))) {

                sources = word(_packetBuffer[0],_packetBuffer[1]);
                if (sources<=20) {
                    #if ATEM_debug
                    temp = atemTallyByIndexSources;
                    #endif
                    atemTallyByIndexSources = word(_packetBuffer[0], _pac
ketBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTallyByIndexSources!=
temp) || (_serialOutput==0x81 && !hasInitialized()))   {
                        Serial.print(F("atemTallyByIndexSources = "));
                        Serial.println(atemTallyByIndexSources);
                    }
                    #endif

                    for(uint8_t a=0;a<sources;a++)  {
                        #if ATEM_debug
                        temp = atemTallyByIndexTallyFlags[a];
                        #endif
```

```
                        atemTallyByIndexTallyFlags[a] = _packetBuffer[2+a
];
                        #if ATEM_debug
                        if ((_serialOutput==0x80 && atemTallyByIndexTally
Flags[a]!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                            Serial.print(F("atemTallyByIndexTallyFlags[a=
")); Serial.print(a); Serial.print(F("] = "));
                            Serial.println(atemTallyByIndexTallyFlags[a])
;
                        }
                        #endif
                    }

                }
            } else
            if(!strcmp(cmdStr, ("TlSr"))) {

                sources = word(_packetBuffer[0],_packetBuffer[1]);
                if (sources<=41) {
                    #if ATEM_debug
                    temp = atemTallyBySourceSources;
                    #endif
                    atemTallyBySourceSources = word(_packetBuffer[0], _pa
cketBuffer[1]);
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemTallyBySourceSources!
=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemTallyBySourceSources = "));
                        Serial.println(atemTallyBySourceSources);
                    }
                    #endif

                    int readComp = 2;
                    for(uint8_t a=0;a<sources;a++)  {
                        if (2+(3*a) == readBytesForTlSr)     {
                            readComp-=readBytesForTlSr;
                            _readToPacketBuffer();
                        }

                        #if ATEM_debug
                        temp = atemTallyBySourceVideoSource[a];
                        #endif
                        atemTallyBySourceVideoSource[a] = word(_packetBuf
fer[readComp+(3*a)], _packetBuffer[readComp+(3*a)+1]);
                        #if ATEM_debug
                        if ((_serialOutput==0x80 && atemTallyBySourceVide
oSource[a]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                            Serial.print(F("atemTallyBySourceVideoSource[
a=")); Serial.print(a); Serial.print(F("] = "));
```

```
                                        Serial.println(atemTallyBySourceVideoSource[a
]);
                                    }
                                    #endif

                                    #if ATEM_debug
                                    temp = atemTallyBySourceTallyFlags[a];
                                    #endif
                                    atemTallyBySourceTallyFlags[a] = _packetBuffer[re
adComp+(3*a)+2];
                                    #if ATEM_debug
                                    if ((_serialOutput==0x80 && atemTallyBySourceTall
yFlags[a]!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                                        Serial.print(F("atemTallyBySourceTallyFlags[a
=")); Serial.print(a); Serial.print(F("] = "));
                                        Serial.println(atemTallyBySourceTallyFlags[a]
);
                                    }
                                    #endif
                                }



                            }
                        } else
                        if(!strcmp(cmdStr, ("Time"))) {

                                #if ATEM_debug
                                temp = atemLastStateChangeTimeCodeHour;
                                #endif
                                atemLastStateChangeTimeCodeHour = _packetBuffer[0];
                                #if ATEM_debug
                                if ((_serialOutput==0x80 && atemLastStateChangeTimeCo
deHour!=temp) || (_serialOutput==0x81 && !hasInitialized()))    {
                                    Serial.print(F("atemLastStateChangeTimeCodeHour =
 "));

                                    Serial.println(atemLastStateChangeTimeCodeHour);
                                }
                                #endif

                                #if ATEM_debug
                                temp = atemLastStateChangeTimeCodeMinute;
                                #endif
                                atemLastStateChangeTimeCodeMinute = _packetBuffer[1];
                                #if ATEM_debug
                                if ((_serialOutput==0x80 && atemLastStateChangeTimeCo
deMinute!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                                    Serial.print(F("atemLastStateChangeTimeCodeMinute
 = "));
```

```
                        Serial.println(atemLastStateChangeTimeCodeMinute)
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemLastStateChangeTimeCodeSecond;
                    #endif
                    atemLastStateChangeTimeCodeSecond = _packetBuffer[2];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemLastStateChangeTimeCo
deSecond!=temp) || (_serialOutput==0x81 && !hasInitialized())) {
                        Serial.print(F("atemLastStateChangeTimeCodeSecond
 = "));

                        Serial.println(atemLastStateChangeTimeCodeSecond)
;
                    }
                    #endif

                    #if ATEM_debug
                    temp = atemLastStateChangeTimeCodeFrame;
                    #endif
                    atemLastStateChangeTimeCodeFrame = _packetBuffer[3];
                    #if ATEM_debug
                    if ((_serialOutput==0x80 && atemLastStateChangeTimeCo
deFrame!=temp) || (_serialOutput==0x81 && !hasInitialized()))  {
                        Serial.print(F("atemLastStateChangeTimeCodeFrame
= "));

                        Serial.println(atemLastStateChangeTimeCodeFrame);
                    }
                    #endif

            } else
            {}
        }




            /**
             * Get Protocol Version; Major
             */
            uint16_t ATEMext::getProtocolVersionMajor() {
                return atemProtocolVersionMajor;
            }

            /**
             * Get Protocol Version; Minor
```

```
 */
uint16_t ATEMext::getProtocolVersionMinor() {
    return atemProtocolVersionMinor;
}

/**
 * Get Product Id; Name
 */
char *  ATEMext::getProductIdName() {
    return atemProductIdName;
}

/**
 * Get Topology; MEs
 */
uint8_t ATEMext::getTopologyMEs() {
    return atemTopologyMEs;
}

/**
 * Get Topology; Sources
 */
uint8_t ATEMext::getTopologySources() {
    return atemTopologySources;
}

/**
 * Get Topology; Color Generators(?)
 */
uint8_t ATEMext::getTopologyColorGenerators() {
    return atemTopologyColorGenerators;
}

/**
 * Get Topology; AUX busses
 */
uint8_t ATEMext::getTopologyAUXbusses() {
    return atemTopologyAUXbusses;
}

/**
 * Get Topology; Downstream Keyes(?)
 */
uint8_t ATEMext::getTopologyDownstreamKeyes() {
    return atemTopologyDownstreamKeyes;
}

/**
 * Get Topology; Stingers(?)
```

```
 */
uint8_t ATEMext::getTopologyStingers() {
    return atemTopologyStingers;
}

/**
 * Get Topology; DVEs(?)
 */
uint8_t ATEMext::getTopologyDVEs() {
    return atemTopologyDVEs;
}

/**
 * Get Topology; SuperSources(?)
 */
uint8_t ATEMext::getTopologySuperSources() {
    return atemTopologySuperSources;
}

/**
 * Get Topology; Has SD Output
 */
bool ATEMext::getTopologyHasSDOutput() {
    return atemTopologyHasSDOutput;
}

/**
 * Get Mix Effect Config; Keyers On ME
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMext::getMixEffectConfigKeyersOnME(uint8_t mE) {
    return atemMixEffectConfigKeyersOnME[mE];
}

/**
 * Get Media Players; Still Banks
 */
uint8_t ATEMext::getMediaPlayersStillBanks() {
    return atemMediaPlayersStillBanks;
}

/**
 * Get Media Players; Clip Banks
 */
uint8_t ATEMext::getMediaPlayersClipBanks() {
    return atemMediaPlayersClipBanks;
}

/**
```

```
 * Get Multi View Config; Multi Viewers
 */
uint8_t ATEMext::getMultiViewConfigMultiViewers() {
    return atemMultiViewConfigMultiViewers;
}

/**
 * Get Super Source Config; Boxes
 */
uint8_t ATEMext::getSuperSourceConfigBoxes() {
    return atemSuperSourceConfigBoxes;
}

/**
 * Get Audio Mixer Config; Audio Channels
 */
uint8_t ATEMext::getAudioMixerConfigAudioChannels() {
    return atemAudioMixerConfigAudioChannels;
}

/**
 * Get Audio Mixer Config; Has Monitor
 */
bool ATEMext::getAudioMixerConfigHasMonitor() {
    return atemAudioMixerConfigHasMonitor;
}

/**
 * Get Video Mixer Config; Modes
 */
uint32_t ATEMext::getVideoMixerConfigModes() {
    return atemVideoMixerConfigModes;
}

/**
 * Get Macro Pool; Banks
 */
uint8_t ATEMext::getMacroPoolBanks() {
    return atemMacroPoolBanks;
}

/**
 * Get Down Converter; Mode
 */
uint8_t ATEMext::getDownConverterMode() {
    return atemDownConverterMode;
}

/**
```

```
 * Set Down Converter; Mode
 * mode     0: Center Cut, 1: Letterbox, 2: Anamorphic
 */
void ATEMext::setDownConverterMode(uint8_t mode) {

    _prepareCommandPacket(("CDcO"),4);

    _packetBuffer[12+_cBBO+4+4+0] = mode;

    _finishCommandPacket();

}

/**
 * Get Video Mode; Format
 */
uint8_t ATEMext::getVideoModeFormat() {
    return atemVideoModeFormat;
}

/**
 * Set Video Mode; Format
 * format   0: 525i59.94 NTSC, 1: 625i 50 PAL, 2: 525i59.94 NTSC 16:9, 3: 625i 50 PAL 16:9, 4: 720p50, 5: 720p59.94, 6: 1080i50, 7: 1080i59.94, 8: 1080p23.98, 9: 1080p24, 10: 1080p25, 11: 1080p29.97, 12: 1080p50, 13: 1080p59.94, 14: 2160p23.98, 15: 2160p24, 16: 2160p25, 17: 2160p29.97
 */
void ATEMext::setVideoModeFormat(uint8_t format) {

    _prepareCommandPacket(("CVdM"),4);

    _packetBuffer[12+_cBBO+4+4+0] = format;

    _finishCommandPacket();

}

/**
 * Get Input Properties; Short Name
 * videoSource  (See video source list)
 */
char *  ATEMext::getInputShortName(uint16_t videoSource) {
    return atemInputShortName[getVideoSrcIndex(videoSource)];
}

/**
 * Get Input Properties; Availability
 * videoSource  (See video source list)
```

```
 */
uint8_t ATEMext::getInputAvailability(uint16_t videoSource) {
    return atemInputAvailability[getVideoSrcIndex(videoSource
)];
}

/**
 * Get Input Properties; ME Availability
 * videoSource  (See video source list)
 */
uint8_t ATEMext::getInputMEAvailability(uint16_t videoSource)
 {
    return atemInputMEAvailability[getVideoSrcIndex(videoSour
ce)];
}

/**
 * Set Input Properties; Long Name
 * videoSource  (See video source list)
 * longName
 */
void ATEMext::setInputLongName(uint16_t videoSource, char *
longName) {

    _prepareCommandPacket(("CInL"),32,(_packetBuffer[12+_cBBO
+4+4+2]==highByte(videoSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lowByt
e(videoSource)));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(videoSource);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(videoSource);

    strncpy((char *)(_packetBuffer+12+4+4+4), longName, 20);

    _finishCommandPacket();

}

/**
 * Set Input Properties; Short Name
 * videoSource  (See video source list)
 * shortName
 */
void ATEMext::setInputShortName(uint16_t videoSource, char *
 shortName) {
```

```
                _prepareCommandPacket(("CInL"),32,(_packetBuffer[12+_cBBO
+4+4+2]==highByte(videoSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lowByt
e(videoSource)));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(videoSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(videoSource);

                strncpy((char *)(_packetBuffer+12+4+4+24), shortName, 4);

                _finishCommandPacket();

            }

            /**
             * Set Input Properties; External Port Type
             * videoSource  (See video source list)
             * externalPortType
             */
            void ATEMext::setInputExternalPortType(uint16_t videoSource,
uint16_t externalPortType) {

                _prepareCommandPacket(("CInL"),32,(_packetBuffer[12+_cBBO
+4+4+2]==highByte(videoSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lowByt
e(videoSource)));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+0] |= 4;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(videoSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(videoSource);

                _packetBuffer[12+_cBBO+4+4+28] = highByte(externalPortTyp
e);
                _packetBuffer[12+_cBBO+4+4+29] = lowByte(externalPortType
);

                _finishCommandPacket();

            }

            /**
             * Set Multi Viewer Properties; Layout
             * multiViewer  0: 1, 1: 2
             * layout   0: Top, 1: Bottom, 2: Left, 3: Right
             */
```

```
		void ATEMext::setMultiViewerPropertiesLayout(uint8_t multiVie
wer, uint8_t layout) {

			_prepareCommandPacket(("CMvP"),4,(_packetBuffer[12+_cBBO+
4+4+1]==multiViewer));

				// Set Mask: 1
			_packetBuffer[12+_cBBO+4+4+0] |= 1;

			_packetBuffer[12+_cBBO+4+4+1] = multiViewer;

			_packetBuffer[12+_cBBO+4+4+2] = layout;

			_finishCommandPacket();

		}

		/**
		 * Get Multi Viewer Input; Video Source
		 * multiViewer  0: 1, 1: 2
		 * windowIndex  0-9
		 */
		uint16_t ATEMext::getMultiViewerInputVideoSource(uint8_t mult
iViewer, uint8_t windowIndex) {
			return atemMultiViewerInputVideoSource[multiViewer][windo
wIndex];
		}

		/**
		 * Set Multi Viewer Input; Video Source
		 * multiViewer  0: 1, 1: 2
		 * windowIndex  2-9
		 * videoSource  (See video source list)
		 */
		void ATEMext::setMultiViewerInputVideoSource(uint8_t multiVie
wer, uint8_t windowIndex, uint16_t videoSource) {

			_prepareCommandPacket(("CMvI"),4,(_packetBuffer[12+_cBBO+
4+4+0]==multiViewer) && (_packetBuffer[12+_cBBO+4+4+1]==windowIndex));

			_packetBuffer[12+_cBBO+4+4+0] = multiViewer;

			_packetBuffer[12+_cBBO+4+4+1] = windowIndex;

			_packetBuffer[12+_cBBO+4+4+2] = highByte(videoSource);
			_packetBuffer[12+_cBBO+4+4+3] = lowByte(videoSource);

			_finishCommandPacket();
```

```
		}

		/**
		 * Get Program Input; Video Source
		 * mE    0: ME1, 1: ME2
		 */
		uint16_t ATEMext::getProgramInputVideoSource(uint8_t mE) {
			return atemProgramInputVideoSource[mE];
		}

		/**
		 * Set Program Input; Video Source
		 * mE    0: ME1, 1: ME2
		 * videoSource  (See video source list)
		 */
		void ATEMext::setProgramInputVideoSource(uint8_t mE, uint16_t videoSource) {

			_prepareCommandPacket(("CPgI"),4,(_packetBuffer[12+_cBBO+4+4+0]==mE));

			_packetBuffer[12+_cBBO+4+4+0] = mE;

			_packetBuffer[12+_cBBO+4+4+2] = highByte(videoSource);
			_packetBuffer[12+_cBBO+4+4+3] = lowByte(videoSource);

			_finishCommandPacket();

		}

		/**
		 * Get Preview Input; Video Source
		 * mE    0: ME1, 1: ME2
		 */
		uint16_t ATEMext::getPreviewInputVideoSource(uint8_t mE) {
			return atemPreviewInputVideoSource[mE];
		}

		/**
		 * Set Preview Input; Video Source
		 * mE    0: ME1, 1: ME2
		 * videoSource  (See video source list)
		 */
		void ATEMext::setPreviewInputVideoSource(uint8_t mE, uint16_t videoSource) {

			_prepareCommandPacket(("CPvI"),4,(_packetBuffer[12+_cBBO+4+4+0]==mE));
```

```
    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(videoSource);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(videoSource);

    _finishCommandPacket();

}

/**
 * Set Cut; M/E
 * mE    0: ME1, 1: ME2
 */
void ATEMext::performCutME(uint8_t mE) {

    _prepareCommandPacket(("DCut"),4);

    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _finishCommandPacket();

}

/**
 * Set Auto; M/E
 * mE    0: ME1, 1: ME2
 */
void ATEMext::performAutoME(uint8_t mE) {

    _prepareCommandPacket(("DAut"),4);

    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _finishCommandPacket();

}

/**
 * Get Transition; Style
 * mE    0: ME1, 1: ME2
 */
uint8_t ATEMext::getTransitionStyle(uint8_t mE) {
    return atemTransitionStyle[mE];
}

/**
 * Get Transition; Next Transition
 * mE    0: ME1, 1: ME2
 */
```

```
uint8_t ATEMext::getTransitionNextTransition(uint8_t mE) {
    return atemTransitionNextTransition[mE];
}

/**
 * Set Transition; Style
 * mE    0: ME1, 1: ME2
 * style    0: Mix, 1: Dip, 2: Wipe, 3: DVE, 4: Sting
 */
void ATEMext::setTransitionStyle(uint8_t mE, uint8_t style) {

    _prepareCommandPacket(("CTTp"),4,(_packetBuffer[12+_cBBO+
4+4+1]==mE));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+1] = mE;

    _packetBuffer[12+_cBBO+4+4+2] = style;

    _finishCommandPacket();

}

/**
 * Set Transition; Next Transition
 * mE    0: ME1, 1: ME2
 * nextTransition    Bit 0: Background=On/Off, Bit 1: Key 1=On
/Off, Bit 2: Key 2=On/Off, Bit 3: Key 3=On/Off, Bit 4: Key 4=On/Off
 */
void ATEMext::setTransitionNextTransition(uint8_t mE, uint8_t
 nextTransition) {

    _prepareCommandPacket(("CTTp"),4,(_packetBuffer[12+_cBBO+
4+4+1]==mE));

        // Set Mask: 2
    _packetBuffer[12+_cBBO+4+4+0] |= 2;

    _packetBuffer[12+_cBBO+4+4+1] = mE;

    _packetBuffer[12+_cBBO+4+4+3] = nextTransition;

    _finishCommandPacket();

}

/**
```

```
 * Get Transition Preview; Enabled
 * mE   0: ME1, 1: ME2
 */
bool ATEMext::getTransitionPreviewEnabled(uint8_t mE) {
    return atemTransitionPreviewEnabled[mE];
}

/**
 * Set Transition Preview; Enabled
 * mE   0: ME1, 1: ME2
 * enabled  Bit 0: On/Off
 */
void ATEMext::setTransitionPreviewEnabled(uint8_t mE, bool en
abled) {

    _prepareCommandPacket(("CTPr"),4,(_packetBuffer[12+_cBBO+
4+4+0]==mE));

    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _packetBuffer[12+_cBBO+4+4+1] = enabled;

    _finishCommandPacket();

}

/**
 * Get Transition Position; In Transition
 * mE   0: ME1, 1: ME2
 */
bool ATEMext::getTransitionInTransition(uint8_t mE) {
    return atemTransitionInTransition[mE];
}

/**
 * Get Transition Position; Frames Remaining
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMext::getTransitionFramesRemaining(uint8_t mE) {
    return atemTransitionFramesRemaining[mE];
}

/**
 * Get Transition Position; Position
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionPosition(uint8_t mE) {
    return atemTransitionPosition[mE];
}
```

```
/**
 * Set Transition Position; Position
 * mE   0: ME1, 1: ME2
 * position     0-9999
 */
void ATEMext::setTransitionPosition(uint8_t mE, uint16_t posi
tion) {

    _prepareCommandPacket(("CTPs"),4,(_packetBuffer[12+_cBBO+
4+4+0]==mE));

    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(position);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(position);

    _finishCommandPacket();

}

/**
 * Get Transition Mix; Rate
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMext::getTransitionMixRate(uint8_t mE) {
    return atemTransitionMixRate[mE];
}

/**
 * Set Transition Mix; Rate
 * mE   0: ME1, 1: ME2
 * rate     1-250: Frames
 */
void ATEMext::setTransitionMixRate(uint8_t mE, uint8_t rate)
{

    _prepareCommandPacket(("CTMx"),4,(_packetBuffer[12+_cBBO+
4+4+0]==mE));

    _packetBuffer[12+_cBBO+4+4+0] = mE;

    _packetBuffer[12+_cBBO+4+4+1] = rate;

    _finishCommandPacket();

}

/**
```

```
 * Get Transition Dip; Rate
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMext::getTransitionDipRate(uint8_t mE) {
    return atemTransitionDipRate[mE];
}

/**
 * Get Transition Dip; Input
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionDipInput(uint8_t mE) {
    return atemTransitionDipInput[mE];
}

/**
 * Set Transition Dip; Rate
 * mE   0: ME1, 1: ME2
 * rate     1-250: Frames
 */
void ATEMext::setTransitionDipRate(uint8_t mE, uint8_t rate)
{

    _prepareCommandPacket(("CTDp"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mE));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+1] = mE;

    _packetBuffer[12+_cBBO+4+4+2] = rate;

    _finishCommandPacket();

}

/**
 * Set Transition Dip; Input
 * mE   0: ME1, 1: ME2
 * input    (See video source list)
 */
void ATEMext::setTransitionDipInput(uint8_t mE, uint16_t inpu
t) {

    _prepareCommandPacket(("CTDp"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mE));

        // Set Mask: 2
```

```
    _packetBuffer[12+_cBBO+4+4+0] |= 2;

    _packetBuffer[12+_cBBO+4+4+1] = mE;

    _packetBuffer[12+_cBBO+4+4+4] = highByte(input);
    _packetBuffer[12+_cBBO+4+4+5] = lowByte(input);

    _finishCommandPacket();

}

/**
 * Get Transition Wipe; Rate
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMext::getTransitionWipeRate(uint8_t mE) {
    return atemTransitionWipeRate[mE];
}

/**
 * Get Transition Wipe; Pattern
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMext::getTransitionWipePattern(uint8_t mE) {
    return atemTransitionWipePattern[mE];
}

/**
 * Get Transition Wipe; Width
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionWipeWidth(uint8_t mE) {
    return atemTransitionWipeWidth[mE];
}

/**
 * Get Transition Wipe; Fill Source
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionWipeFillSource(uint8_t mE) {
    return atemTransitionWipeFillSource[mE];
}

/**
 * Get Transition Wipe; Symmetry
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionWipeSymmetry(uint8_t mE) {
    return atemTransitionWipeSymmetry[mE];
```

```
}

/**
 * Get Transition Wipe; Softness
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionWipeSoftness(uint8_t mE) {
    return atemTransitionWipeSoftness[mE];
}

/**
 * Get Transition Wipe; Position X
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionWipePositionX(uint8_t mE) {
    return atemTransitionWipePositionX[mE];
}

/**
 * Get Transition Wipe; Position Y
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionWipePositionY(uint8_t mE) {
    return atemTransitionWipePositionY[mE];
}

/**
 * Get Transition Wipe; Reverse
 * mE   0: ME1, 1: ME2
 */
bool ATEMext::getTransitionWipeReverse(uint8_t mE) {
    return atemTransitionWipeReverse[mE];
}

/**
 * Get Transition Wipe; FlipFlop
 * mE   0: ME1, 1: ME2
 */
bool ATEMext::getTransitionWipeFlipFlop(uint8_t mE) {
    return atemTransitionWipeFlipFlop[mE];
}

/**
 * Set Transition Wipe; Rate
 * mE   0: ME1, 1: ME2
 * rate     1-250: Frames
 */
void ATEMext::setTransitionWipeRate(uint8_t mE, uint8_t rate)
{
```

```
		_prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

			// Set Mask: 1
		_packetBuffer[12+_cBBO+4+4+1] |= 1;

		_packetBuffer[12+_cBBO+4+4+2] = mE;

		_packetBuffer[12+_cBBO+4+4+3] = rate;

		_finishCommandPacket();

	}

	/**
	 * Set Transition Wipe; Pattern
	 * mE   0: ME1, 1: ME2
	 * pattern  0-17: Pattern Styles
	 */
	void ATEMext::setTransitionWipePattern(uint8_t mE, uint8_t pa
ttern) {

		_prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

			// Set Mask: 2
		_packetBuffer[12+_cBBO+4+4+1] |= 2;

		_packetBuffer[12+_cBBO+4+4+2] = mE;

		_packetBuffer[12+_cBBO+4+4+4] = pattern;

		_finishCommandPacket();

	}

	/**
	 * Set Transition Wipe; Width
	 * mE   0: ME1, 1: ME2
	 * width    0-10000: 0-100%
	 */
	void ATEMext::setTransitionWipeWidth(uint8_t mE, uint16_t wid
th) {

		_prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

			// Set Mask: 4
```

```
				_packetBuffer[12+_cBBO+4+4+1] |= 4;

				_packetBuffer[12+_cBBO+4+4+2] = mE;

				_packetBuffer[12+_cBBO+4+4+6] = highByte(width);
				_packetBuffer[12+_cBBO+4+4+7] = lowByte(width);

				_finishCommandPacket();

			}

			/**
			 * Set Transition Wipe; Fill Source
			 * mE	0: ME1, 1: ME2
			 * fillSource	(See video source list)
			 */
			void ATEMext::setTransitionWipeFillSource(uint8_t mE, uint16_
t fillSource) {

				_prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

					// Set Mask: 8
				_packetBuffer[12+_cBBO+4+4+1] |= 8;

				_packetBuffer[12+_cBBO+4+4+2] = mE;

				_packetBuffer[12+_cBBO+4+4+8] = highByte(fillSource);
				_packetBuffer[12+_cBBO+4+4+9] = lowByte(fillSource);

				_finishCommandPacket();

			}

			/**
			 * Set Transition Wipe; Symmetry
			 * mE	0: ME1, 1: ME2
			 * symmetry	0-10000: 0-100%
			 */
			void ATEMext::setTransitionWipeSymmetry(uint8_t mE, uint16_t
symmetry) {

				_prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

					// Set Mask: 16
				_packetBuffer[12+_cBBO+4+4+1] |= 16;

				_packetBuffer[12+_cBBO+4+4+2] = mE;
```

```
                _packetBuffer[12+_cBBO+4+4+10] = highByte(symmetry);
                _packetBuffer[12+_cBBO+4+4+11] = lowByte(symmetry);

                _finishCommandPacket();

        }

        /**
         * Set Transition Wipe; Softness
         * mE    0: ME1, 1: ME2
         * softness     0-10000: 0-100%
         */
        void ATEMext::setTransitionWipeSoftness(uint8_t mE, uint16_t
softness) {

                _prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                        // Set Mask: 32
                _packetBuffer[12+_cBBO+4+4+1] |= 32;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+12] = highByte(softness);
                _packetBuffer[12+_cBBO+4+4+13] = lowByte(softness);

                _finishCommandPacket();

        }

        /**
         * Set Transition Wipe; Position X
         * mE    0: ME1, 1: ME2
         * positionX    0-10000: 0.0-1.0
         */
        void ATEMext::setTransitionWipePositionX(uint8_t mE, uint16_t
 positionX) {

                _prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                        // Set Mask: 64
                _packetBuffer[12+_cBBO+4+4+1] |= 64;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+14] = highByte(positionX);
                _packetBuffer[12+_cBBO+4+4+15] = lowByte(positionX);
```

```
		_finishCommandPacket();

	}

	/**
	 * Set Transition Wipe; Position Y
	 * mE    0: ME1, 1: ME2
	 * positionY    0-10000: 0.0-1.0
	 */
	void ATEMext::setTransitionWipePositionY(uint8_t mE, uint16_t
 positionY) {

		_prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

			// Set Mask: 128
		_packetBuffer[12+_cBBO+4+4+1] |= 128;

		_packetBuffer[12+_cBBO+4+4+2] = mE;

		_packetBuffer[12+_cBBO+4+4+16] = highByte(positionY);
		_packetBuffer[12+_cBBO+4+4+17] = lowByte(positionY);

		_finishCommandPacket();

	}

	/**
	 * Set Transition Wipe; Reverse
	 * mE    0: ME1, 1: ME2
	 * reverse   Bit 0: On/Off
	 */
	void ATEMext::setTransitionWipeReverse(uint8_t mE, bool rever
se) {

		_prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

			// Set Mask: 256
		_packetBuffer[12+_cBBO+4+4+0] |= 1;

		_packetBuffer[12+_cBBO+4+4+2] = mE;

		_packetBuffer[12+_cBBO+4+4+18] = reverse;

		_finishCommandPacket();

	}
```

```
            /**
             * Set Transition Wipe; FlipFlop
             * mE    0: ME1, 1: ME2
             * flipFlop      Bit 0: On/Off
             */
            void ATEMext::setTransitionWipeFlipFlop(uint8_t mE, bool flip
Flop) {

                _prepareCommandPacket(("CTWp"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                    // Set Mask: 512
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+19] = flipFlop;

                _finishCommandPacket();

            }

            /**
             * Get Transition DVE; Rate
             * mE    0: ME1, 1: ME2
             */
            uint8_t ATEMext::getTransitionDVERate(uint8_t mE) {
                return atemTransitionDVERate[mE];
            }

            /**
             * Get Transition DVE; Style
             * mE    0: ME1, 1: ME2
             */
            uint8_t ATEMext::getTransitionDVEStyle(uint8_t mE) {
                return atemTransitionDVEStyle[mE];
            }

            /**
             * Get Transition DVE; Fill Source
             * mE    0: ME1, 1: ME2
             */
            uint16_t ATEMext::getTransitionDVEFillSource(uint8_t mE) {
                return atemTransitionDVEFillSource[mE];
            }

            /**
             * Get Transition DVE; Key Source
```

```
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionDVEKeySource(uint8_t mE) {
    return atemTransitionDVEKeySource[mE];
}

/**
 * Get Transition DVE; Enable Key
 * mE   0: ME1, 1: ME2
 */
bool ATEMext::getTransitionDVEEnableKey(uint8_t mE) {
    return atemTransitionDVEEnableKey[mE];
}

/**
 * Get Transition DVE; Pre Multiplied
 * mE   0: ME1, 1: ME2
 */
bool ATEMext::getTransitionDVEPreMultiplied(uint8_t mE) {
    return atemTransitionDVEPreMultiplied[mE];
}

/**
 * Get Transition DVE; Clip
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionDVEClip(uint8_t mE) {
    return atemTransitionDVEClip[mE];
}

/**
 * Get Transition DVE; Gain
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionDVEGain(uint8_t mE) {
    return atemTransitionDVEGain[mE];
}

/**
 * Get Transition DVE; Invert Key
 * mE   0: ME1, 1: ME2
 */
bool ATEMext::getTransitionDVEInvertKey(uint8_t mE) {
    return atemTransitionDVEInvertKey[mE];
}

/**
 * Get Transition DVE; Reverse
 * mE   0: ME1, 1: ME2
```

```
 */
bool ATEMext::getTransitionDVEReverse(uint8_t mE) {
    return atemTransitionDVEReverse[mE];
}

/**
 * Get Transition DVE; FlipFlop
 * mE   0: ME1, 1: ME2
 */
bool ATEMext::getTransitionDVEFlipFlop(uint8_t mE) {
    return atemTransitionDVEFlipFlop[mE];
}

/**
 * Set Transition DVE; Rate
 * mE   0: ME1, 1: ME2
 * rate     1-250: Frames
 */
void ATEMext::setTransitionDVERate(uint8_t mE, uint8_t rate)
{

    _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+1] |= 1;

    _packetBuffer[12+_cBBO+4+4+2] = mE;

    _packetBuffer[12+_cBBO+4+4+3] = rate;

    _finishCommandPacket();

}

/**
 * Set Transition DVE; Style
 * mE   0: ME1, 1: ME2
 * style    0-34
 */
void ATEMext::setTransitionDVEStyle(uint8_t mE, uint8_t style
) {

    _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

        // Set Mask: 4
    _packetBuffer[12+_cBBO+4+4+1] |= 4;
```

```
                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = style;

                _finishCommandPacket();

            }

            /**
             * Set Transition DVE; Fill Source
             * mE    0: ME1, 1: ME2
             * fillSource    (See video source list)
             */
            void ATEMext::setTransitionDVEFillSource(uint8_t mE, uint16_t
 fillSource) {

                _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                    // Set Mask: 8
                _packetBuffer[12+_cBBO+4+4+1] |= 8;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+6] = highByte(fillSource);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(fillSource);

                _finishCommandPacket();

            }

            /**
             * Set Transition DVE; Key Source
             * mE    0: ME1, 1: ME2
             * keySource    (See video source list)
             */
            void ATEMext::setTransitionDVEKeySource(uint8_t mE, uint16_t
keySource) {

                _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                    // Set Mask: 16
                _packetBuffer[12+_cBBO+4+4+1] |= 16;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+8] = highByte(keySource);
                _packetBuffer[12+_cBBO+4+4+9] = lowByte(keySource);
```

```
            _finishCommandPacket();

        }

        /**
         * Set Transition DVE; Enable Key
         * mE    0: ME1, 1: ME2
         * enableKey     Bit 0: On/Off
         */
        void ATEMext::setTransitionDVEEnableKey(uint8_t mE, bool enab
leKey) {

            _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 32
            _packetBuffer[12+_cBBO+4+4+1] |= 32;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+10] = enableKey;

            _finishCommandPacket();

        }

        /**
         * Set Transition DVE; Pre Multiplied
         * mE    0: ME1, 1: ME2
         * preMultiplied     Bit 0: On/Off
         */
        void ATEMext::setTransitionDVEPreMultiplied(uint8_t mE, bool
preMultiplied) {

            _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 64
            _packetBuffer[12+_cBBO+4+4+1] |= 64;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+11] = preMultiplied;

            _finishCommandPacket();

        }
```

```
            /**
             * Set Transition DVE; Clip
             * mE    0: ME1, 1: ME2
             * clip      0-1000: 0-100%
             */
            void ATEMext::setTransitionDVEClip(uint8_t mE, uint16_t clip)
 {

                _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                    // Set Mask: 128
                _packetBuffer[12+_cBBO+4+4+1] |= 128;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+12] = highByte(clip);
                _packetBuffer[12+_cBBO+4+4+13] = lowByte(clip);

                _finishCommandPacket();

            }

            /**
             * Set Transition DVE; Gain
             * mE    0: ME1, 1: ME2
             * gain      0-1000: 0-100%
             */
            void ATEMext::setTransitionDVEGain(uint8_t mE, uint16_t gain)
 {

                _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                    // Set Mask: 256
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+14] = highByte(gain);
                _packetBuffer[12+_cBBO+4+4+15] = lowByte(gain);

                _finishCommandPacket();

            }

            /**
             * Set Transition DVE; Invert Key
             * mE    0: ME1, 1: ME2
```

```
 * invertKey    Bit 0: On/Off
 */
void ATEMext::setTransitionDVEInvertKey(uint8_t mE, bool inve
rtKey) {

    _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

        // Set Mask: 512
    _packetBuffer[12+_cBBO+4+4+0] |= 2;

    _packetBuffer[12+_cBBO+4+4+2] = mE;

    _packetBuffer[12+_cBBO+4+4+16] = invertKey;

    _finishCommandPacket();

}

/**
 * Set Transition DVE; Reverse
 * mE    0: ME1, 1: ME2
 * reverse  Bit 0: On/Off
 */
void ATEMext::setTransitionDVEReverse(uint8_t mE, bool revers
e) {

    _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

        // Set Mask: 1024
    _packetBuffer[12+_cBBO+4+4+0] |= 4;

    _packetBuffer[12+_cBBO+4+4+2] = mE;

    _packetBuffer[12+_cBBO+4+4+17] = reverse;

    _finishCommandPacket();

}

/**
 * Set Transition DVE; FlipFlop
 * mE    0: ME1, 1: ME2
 * flipFlop     Bit 0: On/Off
 */
void ATEMext::setTransitionDVEFlipFlop(uint8_t mE, bool flipF
lop) {
```

```
            _prepareCommandPacket(("CTDv"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 2048
            _packetBuffer[12+_cBBO+4+4+0] |= 8;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+18] = flipFlop;

            _finishCommandPacket();

        }

        /**
         * Get Transition Stinger; Source
         * mE    0: ME1, 1: ME2
         */
        uint8_t ATEMext::getTransitionStingerSource(uint8_t mE) {
            return atemTransitionStingerSource[mE];
        }

        /**
         * Get Transition Stinger; Pre Multiplied
         * mE    0: ME1, 1: ME2
         */
        bool ATEMext::getTransitionStingerPreMultiplied(uint8_t mE) {
            return atemTransitionStingerPreMultiplied[mE];
        }

        /**
         * Get Transition Stinger; Clip
         * mE    0: ME1, 1: ME2
         */
        uint16_t ATEMext::getTransitionStingerClip(uint8_t mE) {
            return atemTransitionStingerClip[mE];
        }

        /**
         * Get Transition Stinger; Gain
         * mE    0: ME1, 1: ME2
         */
        uint16_t ATEMext::getTransitionStingerGain(uint8_t mE) {
            return atemTransitionStingerGain[mE];
        }

        /**
         * Get Transition Stinger; Invert Key
         * mE    0: ME1, 1: ME2
```

```
 */
bool ATEMext::getTransitionStingerInvertKey(uint8_t mE) {
    return atemTransitionStingerInvertKey[mE];
}

/**
 * Get Transition Stinger; Pre Roll
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionStingerPreRoll(uint8_t mE) {
    return atemTransitionStingerPreRoll[mE];
}

/**
 * Get Transition Stinger; Clip Duration
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionStingerClipDuration(uint8_t mE
) {
    return atemTransitionStingerClipDuration[mE];
}

/**
 * Get Transition Stinger; Trigger Point
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionStingerTriggerPoint(uint8_t mE
) {
    return atemTransitionStingerTriggerPoint[mE];
}

/**
 * Get Transition Stinger; Mix Rate
 * mE   0: ME1, 1: ME2
 */
uint16_t ATEMext::getTransitionStingerMixRate(uint8_t mE) {
    return atemTransitionStingerMixRate[mE];
}

/**
 * Set Transition Stinger; Source
 * mE   0: ME1, 1: ME2
 * source   1: Media Player 1, 2: Media Player 2
 */
void ATEMext::setTransitionStingerSource(uint8_t mE, uint8_t
source) {

    _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));
```

```
                // Set Mask: 1
            _packetBuffer[12+_cBBO+4+4+1] |= 1;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+3] = source;

            _finishCommandPacket();

        }

        /**
         * Set Transition Stinger; Pre Multiplied
         * mE    0: ME1, 1: ME2
         * preMultiplied    Bit 0: On/Off
         */
        void ATEMext::setTransitionStingerPreMultiplied(uint8_t mE, b
ool preMultiplied) {

            _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 2
            _packetBuffer[12+_cBBO+4+4+1] |= 2;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+4] = preMultiplied;

            _finishCommandPacket();

        }

        /**
         * Set Transition Stinger; Clip
         * mE    0: ME1, 1: ME2
         * clip     0-1000: 0-100%
         */
        void ATEMext::setTransitionStingerClip(uint8_t mE, uint16_t c
lip) {

            _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 4
            _packetBuffer[12+_cBBO+4+4+1] |= 4;

            _packetBuffer[12+_cBBO+4+4+2] = mE;
```

```
            _packetBuffer[12+_cBBO+4+4+6] = highByte(clip);
            _packetBuffer[12+_cBBO+4+4+7] = lowByte(clip);

            _finishCommandPacket();

        }

        /**
         * Set Transition Stinger; Gain
         * mE    0: ME1, 1: ME2
         * gain      0-1000: 0-100%
         */
        void ATEMext::setTransitionStingerGain(uint8_t mE, uint16_t g
ain) {

            _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 8
            _packetBuffer[12+_cBBO+4+4+1] |= 8;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+8] = highByte(gain);
            _packetBuffer[12+_cBBO+4+4+9] = lowByte(gain);

            _finishCommandPacket();

        }

        /**
         * Set Transition Stinger; Invert Key
         * mE    0: ME1, 1: ME2
         * invertKey     Bit 0: On/Off
         */
        void ATEMext::setTransitionStingerInvertKey(uint8_t mE, bool
invertKey) {

            _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 16
            _packetBuffer[12+_cBBO+4+4+1] |= 16;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+10] = invertKey;
```

```
            _finishCommandPacket();

        }

        /**
         * Set Transition Stinger; Pre Roll
         * mE   0: ME1, 1: ME2
         * preRoll  Frames
         */
        void ATEMext::setTransitionStingerPreRoll(uint8_t mE, uint16_
t preRoll) {

            _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 32
            _packetBuffer[12+_cBBO+4+4+1] |= 32;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+12] = highByte(preRoll);
            _packetBuffer[12+_cBBO+4+4+13] = lowByte(preRoll);

            _finishCommandPacket();

        }

        /**
         * Set Transition Stinger; Clip Duration
         * mE   0: ME1, 1: ME2
         * clipDuration     Frames
         */
        void ATEMext::setTransitionStingerClipDuration(uint8_t mE, ui
nt16_t clipDuration) {

            _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                // Set Mask: 64
            _packetBuffer[12+_cBBO+4+4+1] |= 64;

            _packetBuffer[12+_cBBO+4+4+2] = mE;

            _packetBuffer[12+_cBBO+4+4+14] = highByte(clipDuration);
            _packetBuffer[12+_cBBO+4+4+15] = lowByte(clipDuration);

            _finishCommandPacket();

        }
```

```
            /**
             * Set Transition Stinger; Trigger Point
             * mE    0: ME1, 1: ME2
             * triggerPoint     Frames
             */
            void ATEMext::setTransitionStingerTriggerPoint(uint8_t mE, ui
nt16_t triggerPoint) {

                _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                    // Set Mask: 128
                _packetBuffer[12+_cBBO+4+4+1] |= 128;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(triggerPoint);
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(triggerPoint);

                _finishCommandPacket();

            }

            /**
             * Set Transition Stinger; Mix Rate
             * mE    0: ME1, 1: ME2
             * mixRate   Frames
             */
            void ATEMext::setTransitionStingerMixRate(uint8_t mE, uint16_
t mixRate) {

                _prepareCommandPacket(("CTSt"),20,(_packetBuffer[12+_cBBO
+4+4+2]==mE));

                    // Set Mask: 256
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+2] = mE;

                _packetBuffer[12+_cBBO+4+4+18] = highByte(mixRate);
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(mixRate);

                _finishCommandPacket();

            }

            /**
             * Get Keyer On Air; Enabled
```

```
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            bool ATEMext::getKeyerOnAirEnabled(uint8_t mE, uint8_t keyer)
 {
                return atemKeyerOnAirEnabled[mE][keyer];
            }

            /**
             * Set Keyer On Air; Enabled
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * enabled  Bit 0: On/Off
             */
            void ATEMext::setKeyerOnAirEnabled(uint8_t mE, uint8_t keyer,
 bool enabled) {

                _prepareCommandPacket(("CKOn"),4,(_packetBuffer[12+_cBBO+
4+4+0]==mE) && (_packetBuffer[12+_cBBO+4+4+1]==keyer));

                _packetBuffer[12+_cBBO+4+4+0] = mE;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+2] = enabled;

                _finishCommandPacket();

            }

            /**
             * Get Keyer Base; Type
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint8_t ATEMext::getKeyerType(uint8_t mE, uint8_t keyer) {
                return atemKeyerType[mE][keyer];
            }

            /**
             * Get Keyer Base; Fly Enabled
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            bool ATEMext::getKeyerFlyEnabled(uint8_t mE, uint8_t keyer) {
                return atemKeyerFlyEnabled[mE][keyer];
            }

            /**
```

```
 * Get Keyer Base; Fill Source
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
uint16_t ATEMext::getKeyerFillSource(uint8_t mE, uint8_t keye
r) {
    return atemKeyerFillSource[mE][keyer];
}

/**
 * Get Keyer Base; Key Source
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
uint16_t ATEMext::getKeyerKeySource(uint8_t mE, uint8_t keyer
) {
    return atemKeyerKeySource[mE][keyer];
}

/**
 * Get Keyer Base; Masked
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
bool ATEMext::getKeyerMasked(uint8_t mE, uint8_t keyer) {
    return atemKeyerMasked[mE][keyer];
}

/**
 * Get Keyer Base; Top
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
int16_t ATEMext::getKeyerTop(uint8_t mE, uint8_t keyer) {
    return atemKeyerTop[mE][keyer];
}

/**
 * Get Keyer Base; Bottom
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
int16_t ATEMext::getKeyerBottom(uint8_t mE, uint8_t keyer) {
    return atemKeyerBottom[mE][keyer];
}

/**
 * Get Keyer Base; Left
 * mE   0: ME1, 1: ME2
```

```
 * keyer    0-3: Keyer 1-4
 */
int16_t ATEMext::getKeyerLeft(uint8_t mE, uint8_t keyer) {
    return atemKeyerLeft[mE][keyer];
}

/**
 * Get Keyer Base; Right
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
int16_t ATEMext::getKeyerRight(uint8_t mE, uint8_t keyer) {
    return atemKeyerRight[mE][keyer];
}

/**
 * Set Key Type; Type
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 * type     0: Luma, 1: Chroma, 2: Pattern, 3: DVE
 */
void ATEMext::setKeyerType(uint8_t mE, uint8_t keyer, uint8_t
 type) {

    _prepareCommandPacket(("CKTp"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+1] = mE;

    _packetBuffer[12+_cBBO+4+4+2] = keyer;

    _packetBuffer[12+_cBBO+4+4+3] = type;

    _finishCommandPacket();

}

/**
 * Set Key Type; Fly Enabled
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 * flyEnabled   Bit 0: On/Off
 */
void ATEMext::setKeyerFlyEnabled(uint8_t mE, uint8_t keyer, b
ool flyEnabled) {
```

```
                _prepareCommandPacket(("CKTp"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+4] = flyEnabled;

                _finishCommandPacket();

            }

            /**
             * Set Key Mask; Masked
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * masked    Bit 0: On/Off
             */
            void ATEMext::setKeyerMasked(uint8_t mE, uint8_t keyer, bool
masked) {

                _prepareCommandPacket(("CKMs"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+3] = masked;

                _finishCommandPacket();

            }

            /**
             * Set Key Mask; Top
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * top   -9000-9000: -9.00-9.00
             */
            void ATEMext::setKeyerTop(uint8_t mE, uint8_t keyer, int16_t
top) {
```

```
                _prepareCommandPacket(("CKMs"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(top);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(top);

                _finishCommandPacket();

            }

            /**
             * Set Key Mask; Bottom
             * mE    0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * bottom   -9000-9000: -9.00-9.00
             */
            void ATEMext::setKeyerBottom(uint8_t mE, uint8_t keyer, int16
_t bottom) {

                _prepareCommandPacket(("CKMs"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+0] |= 4;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+6] = highByte(bottom);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(bottom);

                _finishCommandPacket();

            }

            /**
             * Set Key Mask; Left
             * mE    0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * left     -16000-16000: -9.00-9.00
```

```
            */
            void ATEMext::setKeyerLeft(uint8_t mE, uint8_t keyer, int16_t
 left) {

                _prepareCommandPacket(("CKMs"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 8
                _packetBuffer[12+_cBBO+4+4+0] |= 8;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+8] = highByte(left);
                _packetBuffer[12+_cBBO+4+4+9] = lowByte(left);

                _finishCommandPacket();

            }

            /**
             * Set Key Mask; Right
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * right    -16000-16000: -9.00-9.00
             */
            void ATEMext::setKeyerRight(uint8_t mE, uint8_t keyer, int16_
t right) {

                _prepareCommandPacket(("CKMs"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 16
                _packetBuffer[12+_cBBO+4+4+0] |= 16;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+10] = highByte(right);
                _packetBuffer[12+_cBBO+4+4+11] = lowByte(right);

                _finishCommandPacket();

            }

            /**
             * Set Key Fill; Fill Source
```

```
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * fillSource   (See video source list)
             */
            void ATEMext::setKeyerFillSource(uint8_t mE, uint8_t keyer, u
int16_t fillSource) {

                _prepareCommandPacket(("CKeF"),4,(_packetBuffer[12+_cBBO+
4+4+0]==mE) && (_packetBuffer[12+_cBBO+4+4+1]==keyer));

                _packetBuffer[12+_cBBO+4+4+0] = mE;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(fillSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(fillSource);

                _finishCommandPacket();

            }

            /**
             * Set Key Cut; Key Source
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * keySource    (See video source list)
             */
            void ATEMext::setKeyerKeySource(uint8_t mE, uint8_t keyer, ui
nt16_t keySource) {

                _prepareCommandPacket(("CKeC"),4,(_packetBuffer[12+_cBBO+
4+4+0]==mE) && (_packetBuffer[12+_cBBO+4+4+1]==keyer));

                _packetBuffer[12+_cBBO+4+4+0] = mE;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(keySource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(keySource);

                _finishCommandPacket();

            }

            /**
             * Get Key Luma; Pre Multiplied
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
```

```
            bool ATEMext::getKeyLumaPreMultiplied(uint8_t mE, uint8_t key
er) {
                return atemKeyLumaPreMultiplied[mE][keyer];
            }

            /**
             * Get Key Luma; Clip
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyLumaClip(uint8_t mE, uint8_t keyer) {
                return atemKeyLumaClip[mE][keyer];
            }

            /**
             * Get Key Luma; Gain
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyLumaGain(uint8_t mE, uint8_t keyer) {
                return atemKeyLumaGain[mE][keyer];
            }

            /**
             * Get Key Luma; Invert Key
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             */
            bool ATEMext::getKeyLumaInvertKey(uint8_t mE, uint8_t keyer)
{
                return atemKeyLumaInvertKey[mE][keyer];
            }

            /**
             * Set Key Luma; Pre Multiplied
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * preMultiplied     Bit 0: On/Off
             */
            void ATEMext::setKeyLumaPreMultiplied(uint8_t mE, uint8_t key
er, bool preMultiplied) {

                _prepareCommandPacket(("CKLm"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = mE;
```

```
                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+3] = preMultiplied;

                _finishCommandPacket();

            }

            /**
             * Set Key Luma; Clip
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * clip      0-1000: 0-100%
             */
            void ATEMext::setKeyLumaClip(uint8_t mE, uint8_t keyer, uint1
6_t clip) {

                _prepareCommandPacket(("CKLm"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(clip);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(clip);

                _finishCommandPacket();

            }

            /**
             * Set Key Luma; Gain
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * gain      0-1000: 0-100%
             */
            void ATEMext::setKeyLumaGain(uint8_t mE, uint8_t keyer, uint1
6_t gain) {

                _prepareCommandPacket(("CKLm"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+0] |= 4;
```

```
		_packetBuffer[12+_cBBO+4+4+1] = mE;

		_packetBuffer[12+_cBBO+4+4+2] = keyer;

		_packetBuffer[12+_cBBO+4+4+6] = highByte(gain);
		_packetBuffer[12+_cBBO+4+4+7] = lowByte(gain);

		_finishCommandPacket();

	}

	/**
	 * Set Key Luma; Invert Key
	 * mE    0: ME1, 1: ME2
	 * keyer     0-3: Keyer 1-4
	 * invertKey     Bit 0: On/Off
	 */
	void ATEMext::setKeyLumaInvertKey(uint8_t mE, uint8_t keyer, 
bool invertKey) {

		_prepareCommandPacket(("CKLm"),12,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

			// Set Mask: 8
		_packetBuffer[12+_cBBO+4+4+0] |= 8;

		_packetBuffer[12+_cBBO+4+4+1] = mE;

		_packetBuffer[12+_cBBO+4+4+2] = keyer;

		_packetBuffer[12+_cBBO+4+4+8] = invertKey;

		_finishCommandPacket();

	}

	/**
	 * Get Key Chroma; Hue
	 * mE    0: ME1, 1: ME2
	 * keyer     0-3: Keyer 1-4
	 */
	uint16_t ATEMext::getKeyChromaHue(uint8_t mE, uint8_t keyer) 
{
		return atemKeyChromaHue[mE][keyer];
	}

	/**
	 * Get Key Chroma; Gain
```

```
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyChromaGain(uint8_t mE, uint8_t keyer)
 {
                return atemKeyChromaGain[mE][keyer];
            }

            /**
             * Get Key Chroma; Y Suppress
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyChromaYSuppress(uint8_t mE, uint8_t k
eyer) {
                return atemKeyChromaYSuppress[mE][keyer];
            }

            /**
             * Get Key Chroma; Lift
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyChromaLift(uint8_t mE, uint8_t keyer)
 {
                return atemKeyChromaLift[mE][keyer];
            }

            /**
             * Get Key Chroma; Narrow
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            bool ATEMext::getKeyChromaNarrow(uint8_t mE, uint8_t keyer) {
                return atemKeyChromaNarrow[mE][keyer];
            }

            /**
             * Set Key Chroma; Hue
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * hue  0-3599: 0-359.9 Degrees
             */
            void ATEMext::setKeyChromaHue(uint8_t mE, uint8_t keyer, uint
16_t hue) {

                _prepareCommandPacket(("CKCk"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));
```

```
            // Set Mask: 1
        _packetBuffer[12+_cBBO+4+4+0] |= 1;

        _packetBuffer[12+_cBBO+4+4+1] = mE;

        _packetBuffer[12+_cBBO+4+4+2] = keyer;

        _packetBuffer[12+_cBBO+4+4+4] = highByte(hue);
        _packetBuffer[12+_cBBO+4+4+5] = lowByte(hue);

        _finishCommandPacket();

    }

    /**
     * Set Key Chroma; Gain
     * mE   0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     * gain     0-1000: 0-100%
     */
    void ATEMext::setKeyChromaGain(uint8_t mE, uint8_t keyer, uin
t16_t gain) {

        _prepareCommandPacket(("CKCk"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

            // Set Mask: 2
        _packetBuffer[12+_cBBO+4+4+0] |= 2;

        _packetBuffer[12+_cBBO+4+4+1] = mE;

        _packetBuffer[12+_cBBO+4+4+2] = keyer;

        _packetBuffer[12+_cBBO+4+4+6] = highByte(gain);
        _packetBuffer[12+_cBBO+4+4+7] = lowByte(gain);

        _finishCommandPacket();

    }

    /**
     * Set Key Chroma; Y Suppress
     * mE   0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     * ySuppress    0-1000: 0-100%
     */
    void ATEMext::setKeyChromaYSuppress(uint8_t mE, uint8_t keyer
, uint16_t ySuppress) {
```

```
			_prepareCommandPacket(("CKCk"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

				// Set Mask: 4
			_packetBuffer[12+_cBBO+4+4+0] |= 4;

			_packetBuffer[12+_cBBO+4+4+1] = mE;

			_packetBuffer[12+_cBBO+4+4+2] = keyer;

			_packetBuffer[12+_cBBO+4+4+8] = highByte(ySuppress);
			_packetBuffer[12+_cBBO+4+4+9] = lowByte(ySuppress);

			_finishCommandPacket();

		}

		/**
		 * Set Key Chroma; Lift
		 * mE	0: ME1, 1: ME2
		 * keyer	0-3: Keyer 1-4
		 * lift	0-1000: 0-100%
		 */
		void ATEMext::setKeyChromaLift(uint8_t mE, uint8_t keyer, uin
t16_t lift) {

			_prepareCommandPacket(("CKCk"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

				// Set Mask: 8
			_packetBuffer[12+_cBBO+4+4+0] |= 8;

			_packetBuffer[12+_cBBO+4+4+1] = mE;

			_packetBuffer[12+_cBBO+4+4+2] = keyer;

			_packetBuffer[12+_cBBO+4+4+10] = highByte(lift);
			_packetBuffer[12+_cBBO+4+4+11] = lowByte(lift);

			_finishCommandPacket();

		}

		/**
		 * Set Key Chroma; Narrow
		 * mE	0: ME1, 1: ME2
		 * keyer	0-3: Keyer 1-4
		 * narrow	Bit 0: On/Off
		 */
```

```
            void ATEMext::setKeyChromaNarrow(uint8_t mE, uint8_t keyer, b
ool narrow) {

                _prepareCommandPacket(("CKCk"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 16
                _packetBuffer[12+_cBBO+4+4+0] |= 16;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+12] = narrow;

                _finishCommandPacket();

            }

            /**
             * Get Key Pattern; Pattern
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint8_t ATEMext::getKeyPatternPattern(uint8_t mE, uint8_t key
er) {
                return atemKeyPatternPattern[mE][keyer];
            }

            /**
             * Get Key Pattern; Size
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyPatternSize(uint8_t mE, uint8_t keyer
) {
                return atemKeyPatternSize[mE][keyer];
            }

            /**
             * Get Key Pattern; Symmetry
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyPatternSymmetry(uint8_t mE, uint8_t k
eyer) {
                return atemKeyPatternSymmetry[mE][keyer];
            }
```

```
/**
 * Get Key Pattern; Softness
 * mE    0: ME1, 1: ME2
 * keyer     0-3: Keyer 1-4
 */
uint16_t ATEMext::getKeyPatternSoftness(uint8_t mE, uint8_t k
eyer) {
    return atemKeyPatternSoftness[mE][keyer];
}

/**
 * Get Key Pattern; Position X
 * mE    0: ME1, 1: ME2
 * keyer     0-3: Keyer 1-4
 */
uint16_t ATEMext::getKeyPatternPositionX(uint8_t mE, uint8_t
keyer) {
    return atemKeyPatternPositionX[mE][keyer];
}

/**
 * Get Key Pattern; Position Y
 * mE    0: ME1, 1: ME2
 * keyer     0-3: Keyer 1-4
 */
uint16_t ATEMext::getKeyPatternPositionY(uint8_t mE, uint8_t
keyer) {
    return atemKeyPatternPositionY[mE][keyer];
}

/**
 * Get Key Pattern; Invert Pattern
 * mE    0: ME1, 1: ME2
 * keyer     0-3: Keyer 1-4
 */
bool ATEMext::getKeyPatternInvertPattern(uint8_t mE, uint8_t
keyer) {
    return atemKeyPatternInvertPattern[mE][keyer];
}

/**
 * Set Key Pattern; Pattern
 * mE    0: ME1, 1: ME2
 * keyer     0-3: Keyer 1-4
 * pattern   0-17: Pattern Styles
 */
void ATEMext::setKeyPatternPattern(uint8_t mE, uint8_t keyer,
 uint8_t pattern) {
```

```
                _prepareCommandPacket(("CKPt"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+3] = pattern;

                _finishCommandPacket();

            }

            /**
             * Set Key Pattern; Size
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * size     0-10000: 0-100%
             */
            void ATEMext::setKeyPatternSize(uint8_t mE, uint8_t keyer, ui
nt16_t size) {

                _prepareCommandPacket(("CKPt"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(size);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(size);

                _finishCommandPacket();

            }

            /**
             * Set Key Pattern; Symmetry
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * symmetry     0-10000: 0-100%
             */
```

```
void ATEMext::setKeyPatternSymmetry(uint8_t mE, uint8_t keyer
, uint16_t symmetry) {

    _prepareCommandPacket(("CKPt"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

        // Set Mask: 4
    _packetBuffer[12+_cBBO+4+4+0] |= 4;

    _packetBuffer[12+_cBBO+4+4+1] = mE;

    _packetBuffer[12+_cBBO+4+4+2] = keyer;

    _packetBuffer[12+_cBBO+4+4+6] = highByte(symmetry);
    _packetBuffer[12+_cBBO+4+4+7] = lowByte(symmetry);

    _finishCommandPacket();

}

/**
 * Set Key Pattern; Softness
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 * softness     0-10000: 0-100%
 */
void ATEMext::setKeyPatternSoftness(uint8_t mE, uint8_t keyer
, uint16_t softness) {

    _prepareCommandPacket(("CKPt"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

        // Set Mask: 8
    _packetBuffer[12+_cBBO+4+4+0] |= 8;

    _packetBuffer[12+_cBBO+4+4+1] = mE;

    _packetBuffer[12+_cBBO+4+4+2] = keyer;

    _packetBuffer[12+_cBBO+4+4+8] = highByte(softness);
    _packetBuffer[12+_cBBO+4+4+9] = lowByte(softness);

    _finishCommandPacket();

}

/**
 * Set Key Pattern; Position X
 * mE   0: ME1, 1: ME2
```

```
             * keyer    0-3: Keyer 1-4
             * positionX    0-10000: 0.0-1.0
             */
            void ATEMext::setKeyPatternPositionX(uint8_t mE, uint8_t keye
r, uint16_t positionX) {

                _prepareCommandPacket(("CKPt"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 16
                _packetBuffer[12+_cBBO+4+4+0] |= 16;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+10] = highByte(positionX);
                _packetBuffer[12+_cBBO+4+4+11] = lowByte(positionX);

                _finishCommandPacket();

            }

            /**
             * Set Key Pattern; Position Y
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * positionY    0-10000: 0.0-1.0
             */
            void ATEMext::setKeyPatternPositionY(uint8_t mE, uint8_t keye
r, uint16_t positionY) {

                _prepareCommandPacket(("CKPt"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 32
                _packetBuffer[12+_cBBO+4+4+0] |= 32;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+12] = highByte(positionY);
                _packetBuffer[12+_cBBO+4+4+13] = lowByte(positionY);

                _finishCommandPacket();

            }
```

```
            /**
             * Set Key Pattern; Invert Pattern
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * invertPattern     Bit 0: On/Off
             */
            void ATEMext::setKeyPatternInvertPattern(uint8_t mE, uint8_t
keyer, bool invertPattern) {

                _prepareCommandPacket(("CKPt"),16,(_packetBuffer[12+_cBBO
+4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 64
                _packetBuffer[12+_cBBO+4+4+0] |= 64;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+14] = invertPattern;

                _finishCommandPacket();

            }

            /**
             * Get Key DVE; Size X
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             */
            int32_t ATEMext::getKeyDVESizeX(uint8_t mE, uint8_t keyer) {
                return atemKeyDVESizeX[mE][keyer];
            }

            /**
             * Get Key DVE; Size Y
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             */
            int32_t ATEMext::getKeyDVESizeY(uint8_t mE, uint8_t keyer) {
                return atemKeyDVESizeY[mE][keyer];
            }

            /**
             * Get Key DVE; Position X
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             */
```

```
int32_t ATEMext::getKeyDVEPositionX(uint8_t mE, uint8_t keyer
) {
    return atemKeyDVEPositionX[mE][keyer];
}

/**
 * Get Key DVE; Position Y
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
int32_t ATEMext::getKeyDVEPositionY(uint8_t mE, uint8_t keyer
) {
    return atemKeyDVEPositionY[mE][keyer];
}

/**
 * Get Key DVE; Rotation
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
int32_t ATEMext::getKeyDVERotation(uint8_t mE, uint8_t keyer)
 {
    return atemKeyDVERotation[mE][keyer];
}

/**
 * Get Key DVE; Border Enabled
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
bool ATEMext::getKeyDVEBorderEnabled(uint8_t mE, uint8_t keye
r) {
    return atemKeyDVEBorderEnabled[mE][keyer];
}

/**
 * Get Key DVE; Shadow
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
bool ATEMext::getKeyDVEShadow(uint8_t mE, uint8_t keyer) {
    return atemKeyDVEShadow[mE][keyer];
}

/**
 * Get Key DVE; Border Bevel
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
```

```
uint8_t ATEMext::getKeyDVEBorderBevel(uint8_t mE, uint8_t key
er) {
        return atemKeyDVEBorderBevel[mE][keyer];
    }

    /**
     * Get Key DVE; Border Outer Width
     * mE    0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     */
    uint16_t ATEMext::getKeyDVEBorderOuterWidth(uint8_t mE, uint8
_t keyer) {
        return atemKeyDVEBorderOuterWidth[mE][keyer];
    }

    /**
     * Get Key DVE; Border Inner Width
     * mE    0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     */
    uint16_t ATEMext::getKeyDVEBorderInnerWidth(uint8_t mE, uint8
_t keyer) {
        return atemKeyDVEBorderInnerWidth[mE][keyer];
    }

    /**
     * Get Key DVE; Border Outer Softness
     * mE    0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     */
    uint8_t ATEMext::getKeyDVEBorderOuterSoftness(uint8_t mE, uin
t8_t keyer) {
        return atemKeyDVEBorderOuterSoftness[mE][keyer];
    }

    /**
     * Get Key DVE; Border Inner Softness
     * mE    0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     */
    uint8_t ATEMext::getKeyDVEBorderInnerSoftness(uint8_t mE, uin
t8_t keyer) {
        return atemKeyDVEBorderInnerSoftness[mE][keyer];
    }

    /**
     * Get Key DVE; Border Bevel Softness
     * mE    0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
```

```
             */
            uint8_t ATEMext::getKeyDVEBorderBevelSoftness(uint8_t mE, uin
t8_t keyer) {
                return atemKeyDVEBorderBevelSoftness[mE][keyer];
            }

            /**
             * Get Key DVE; Border Bevel Position
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint8_t ATEMext::getKeyDVEBorderBevelPosition(uint8_t mE, uin
t8_t keyer) {
                return atemKeyDVEBorderBevelPosition[mE][keyer];
            }

            /**
             * Get Key DVE; Border Opacity
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint8_t ATEMext::getKeyDVEBorderOpacity(uint8_t mE, uint8_t k
eyer) {
                return atemKeyDVEBorderOpacity[mE][keyer];
            }

            /**
             * Get Key DVE; Border Hue
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyDVEBorderHue(uint8_t mE, uint8_t keye
r) {
                return atemKeyDVEBorderHue[mE][keyer];
            }

            /**
             * Get Key DVE; Border Saturation
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             */
            uint16_t ATEMext::getKeyDVEBorderSaturation(uint8_t mE, uint8
_t keyer) {
                return atemKeyDVEBorderSaturation[mE][keyer];
            }

            /**
             * Get Key DVE; Border Luma
             * mE   0: ME1, 1: ME2
```

```
 * keyer    0-3: Keyer 1-4
 */
uint16_t ATEMext::getKeyDVEBorderLuma(uint8_t mE, uint8_t key
er) {
    return atemKeyDVEBorderLuma[mE][keyer];
}

/**
 * Get Key DVE; Light Source Direction
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
uint16_t ATEMext::getKeyDVELightSourceDirection(uint8_t mE, u
int8_t keyer) {
    return atemKeyDVELightSourceDirection[mE][keyer];
}

/**
 * Get Key DVE; Light Source Altitude
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
uint8_t ATEMext::getKeyDVELightSourceAltitude(uint8_t mE, uin
t8_t keyer) {
    return atemKeyDVELightSourceAltitude[mE][keyer];
}

/**
 * Get Key DVE; Masked
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
bool ATEMext::getKeyDVEMasked(uint8_t mE, uint8_t keyer) {
    return atemKeyDVEMasked[mE][keyer];
}

/**
 * Get Key DVE; Top
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
int16_t ATEMext::getKeyDVETop(uint8_t mE, uint8_t keyer) {
    return atemKeyDVETop[mE][keyer];
}

/**
 * Get Key DVE; Bottom
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
```

```
 */
int16_t ATEMext::getKeyDVEBottom(uint8_t mE, uint8_t keyer) {
    return atemKeyDVEBottom[mE][keyer];
}

/**
 * Get Key DVE; Left
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
int16_t ATEMext::getKeyDVELeft(uint8_t mE, uint8_t keyer) {
    return atemKeyDVELeft[mE][keyer];
}

/**
 * Get Key DVE; Right
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
int16_t ATEMext::getKeyDVERight(uint8_t mE, uint8_t keyer) {
    return atemKeyDVERight[mE][keyer];
}

/**
 * Get Key DVE; Rate
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 */
uint8_t ATEMext::getKeyDVERate(uint8_t mE, uint8_t keyer) {
    return atemKeyDVERate[mE][keyer];
}

/**
 * Set Key DVE; Size X
 * mE   0: ME1, 1: ME2
 * keyer    0-3: Keyer 1-4
 * sizeX    Example: 1000: 1.000
 */
void ATEMext::setKeyDVESizeX(uint8_t mE, uint8_t keyer, int32
_t sizeX) {

    _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+3] |= 1;

    _packetBuffer[12+_cBBO+4+4+4] = mE;
```

```
                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+8] = (int32_t)((sizeX>>24) & 0
xFF);
                _packetBuffer[12+_cBBO+4+4+9] = (int32_t)((sizeX>>16) & 0
xFF);
                _packetBuffer[12+_cBBO+4+4+10] = (int32_t)((sizeX>>8) & 0
xFF);
                _packetBuffer[12+_cBBO+4+4+11] = (int32_t)(sizeX & 0xFF);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Size Y
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * sizeY    Example: 2000: 2.000
             */
            void ATEMext::setKeyDVESizeY(uint8_t mE, uint8_t keyer, int32
_t sizeY) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+3] |= 2;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+12] = (int32_t)((sizeY>>24) &
0xFF);
                _packetBuffer[12+_cBBO+4+4+13] = (int32_t)((sizeY>>16) &
0xFF);
                _packetBuffer[12+_cBBO+4+4+14] = (int32_t)((sizeY>>8) & 0
xFF);
                _packetBuffer[12+_cBBO+4+4+15] = (int32_t)(sizeY & 0xFF);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Position X
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
```

```
             * positionX    Example: 1000: 1.000
             */
            void ATEMext::setKeyDVEPositionX(uint8_t mE, uint8_t keyer, i
nt32_t positionX) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+3] |= 4;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+16] = (int32_t)((positionX>>24
) & 0xFF);
                _packetBuffer[12+_cBBO+4+4+17] = (int32_t)((positionX>>16
) & 0xFF);
                _packetBuffer[12+_cBBO+4+4+18] = (int32_t)((positionX>>8)
 & 0xFF);
                _packetBuffer[12+_cBBO+4+4+19] = (int32_t)(positionX & 0x
FF);


                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Position Y
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * positionY    Example: -1000: -1.000
             */
            void ATEMext::setKeyDVEPositionY(uint8_t mE, uint8_t keyer, i
nt32_t positionY) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 8
                _packetBuffer[12+_cBBO+4+4+3] |= 8;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+20] = (int32_t)((positionY>>24
) & 0xFF);
```

```
                _packetBuffer[12+_cBBO+4+4+21] = (int32_t)((positionY>>16
) & 0xFF);
                _packetBuffer[12+_cBBO+4+4+22] = (int32_t)((positionY>>8)
 & 0xFF);
                _packetBuffer[12+_cBBO+4+4+23] = (int32_t)(positionY & 0x
FF);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Rotation
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * rotation     Example: 3670: 1 rotation+7 degress
             */
            void ATEMext::setKeyDVERotation(uint8_t mE, uint8_t keyer, in
t32_t rotation) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 16
                _packetBuffer[12+_cBBO+4+4+3] |= 16;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+24] = (int32_t)((rotation>>24)
 & 0xFF);
                _packetBuffer[12+_cBBO+4+4+25] = (int32_t)((rotation>>16)
 & 0xFF);
                _packetBuffer[12+_cBBO+4+4+26] = (int32_t)((rotation>>8)
& 0xFF);
                _packetBuffer[12+_cBBO+4+4+27] = (int32_t)(rotation & 0xF
F);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Enabled
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * borderEnabled    Bit 0: On/Off
             */
```

```
            void ATEMext::setKeyDVEBorderEnabled(uint8_t mE, uint8_t keye
r, bool borderEnabled) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 32
                _packetBuffer[12+_cBBO+4+4+3] |= 32;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+28] = borderEnabled;

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Shadow
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * shadow   Bit 0: On/Off
             */
            void ATEMext::setKeyDVEShadow(uint8_t mE, uint8_t keyer, bool
 shadow) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 64
                _packetBuffer[12+_cBBO+4+4+3] |= 64;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+29] = shadow;

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Bevel
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * borderBevel  0: No, 1: In/Out, 2: In, 3: Out
```

```
         */
        void ATEMext::setKeyDVEBorderBevel(uint8_t mE, uint8_t keyer,
 uint8_t borderBevel) {

            _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 128
            _packetBuffer[12+_cBBO+4+4+3] |= 128;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+30] = borderBevel;

            _finishCommandPacket();

        }

        /**
         * Set Key DVE; Border Outer Width
         * mE   0: ME1, 1: ME2
         * keyer    0-3: Keyer 1-4
         * borderOuterWidth     0-1600: 0-16.00
         */
        void ATEMext::setKeyDVEBorderOuterWidth(uint8_t mE, uint8_t k
eyer, uint16_t borderOuterWidth) {

            _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                // Set Mask: 256
            _packetBuffer[12+_cBBO+4+4+2] |= 1;

            _packetBuffer[12+_cBBO+4+4+4] = mE;

            _packetBuffer[12+_cBBO+4+4+5] = keyer;

            _packetBuffer[12+_cBBO+4+4+32] = highByte(borderOuterWidt
h);
            _packetBuffer[12+_cBBO+4+4+33] = lowByte(borderOuterWidth
);

            _finishCommandPacket();

        }

        /**
```

```
             * Set Key DVE; Border Inner Width
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * borderInnerWidth     0-1600: 0-16.00
             */
            void ATEMext::setKeyDVEBorderInnerWidth(uint8_t mE, uint8_t k
eyer, uint16_t borderInnerWidth) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 512
                _packetBuffer[12+_cBBO+4+4+2] |= 2;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+34] = highByte(borderInnerWidt
h);
                _packetBuffer[12+_cBBO+4+4+35] = lowByte(borderInnerWidth
);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Outer Softness
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * borderOuterSoftness  0-100: 0-100%
             */
            void ATEMext::setKeyDVEBorderOuterSoftness(uint8_t mE, uint8_
t keyer, uint8_t borderOuterSoftness) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 1024
                _packetBuffer[12+_cBBO+4+4+2] |= 4;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+36] = borderOuterSoftness;

                _finishCommandPacket();
```

```
            }

            /**
             * Set Key DVE; Border Inner Softness
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * borderInnerSoftness  0-100: 0-100%
             */
            void ATEMext::setKeyDVEBorderInnerSoftness(uint8_t mE, uint8_
t keyer, uint8_t borderInnerSoftness) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 2048
                _packetBuffer[12+_cBBO+4+4+2] |= 8;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+37] = borderInnerSoftness;

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Bevel Softness
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * borderBevelSoftness  0-100: 0.0-1.0
             */
            void ATEMext::setKeyDVEBorderBevelSoftness(uint8_t mE, uint8_
t keyer, uint8_t borderBevelSoftness) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 4096
                _packetBuffer[12+_cBBO+4+4+2] |= 16;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+38] = borderBevelSoftness;
```

```
                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Bevel Position
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * borderBevelPosition  0-100: 0.0-1.0
             */
            void ATEMext::setKeyDVEBorderBevelPosition(uint8_t mE, uint8_
t keyer, uint8_t borderBevelPosition) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 8192
                _packetBuffer[12+_cBBO+4+4+2] |= 32;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+39] = borderBevelPosition;

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Opacity
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * borderOpacity    0-100: 0-100%
             */
            void ATEMext::setKeyDVEBorderOpacity(uint8_t mE, uint8_t keye
r, uint8_t borderOpacity) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 16384
                _packetBuffer[12+_cBBO+4+4+2] |= 64;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+40] = borderOpacity;
```

```
                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Hue
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * borderHue     0-3599: 0-359.9 Degrees
             */
            void ATEMext::setKeyDVEBorderHue(uint8_t mE, uint8_t keyer, u
int16_t borderHue) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 32768
                _packetBuffer[12+_cBBO+4+4+2] |= 128;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+42] = highByte(borderHue);
                _packetBuffer[12+_cBBO+4+4+43] = lowByte(borderHue);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Saturation
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * borderSaturation      0-1000: 0-100%
             */
            void ATEMext::setKeyDVEBorderSaturation(uint8_t mE, uint8_t k
eyer, uint16_t borderSaturation) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 65536
                _packetBuffer[12+_cBBO+4+4+1] |= 1;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;
```

```
                _packetBuffer[12+_cBBO+4+4+44] = highByte(borderSaturatio
n);
                _packetBuffer[12+_cBBO+4+4+45] = lowByte(borderSaturation
);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Border Luma
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * borderLuma    0-1000: 0-100%
             */
            void ATEMext::setKeyDVEBorderLuma(uint8_t mE, uint8_t keyer,
uint16_t borderLuma) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 131072
                _packetBuffer[12+_cBBO+4+4+1] |= 2;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+46] = highByte(borderLuma);
                _packetBuffer[12+_cBBO+4+4+47] = lowByte(borderLuma);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Light Source Direction
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * lightSourceDirection     0-3590: 0-359 Degrees
             */
            void ATEMext::setKeyDVELightSourceDirection(uint8_t mE, uint8
_t keyer, uint16_t lightSourceDirection) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 262144
```

```
                _packetBuffer[12+_cBBO+4+4+1] |= 4;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+48] = highByte(lightSourceDire
ction);
                _packetBuffer[12+_cBBO+4+4+49] = lowByte(lightSourceDirec
tion);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Light Source Altitude
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * lightSourceAltitude  10-100: 10-100
             */
            void ATEMext::setKeyDVELightSourceAltitude(uint8_t mE, uint8_
t keyer, uint8_t lightSourceAltitude) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 524288
                _packetBuffer[12+_cBBO+4+4+1] |= 8;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+50] = lightSourceAltitude;

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Masked
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * masked   Bit 0: On/Off
             */
            void ATEMext::setKeyDVEMasked(uint8_t mE, uint8_t keyer, bool
 masked) {
```

```
                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 1048576
                _packetBuffer[12+_cBBO+4+4+1] |= 16;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+51] = masked;

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Top
             * mE    0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * top  -9000-9000: -9.00-9.00
             */
            void ATEMext::setKeyDVETop(uint8_t mE, uint8_t keyer, int16_t
 top) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 2097152
                _packetBuffer[12+_cBBO+4+4+1] |= 32;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+52] = highByte(top);
                _packetBuffer[12+_cBBO+4+4+53] = lowByte(top);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Bottom
             * mE    0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * bottom    -9000-9000: -9.00-9.00
             */
```

```
            void ATEMext::setKeyDVEBottom(uint8_t mE, uint8_t keyer, int1
6_t bottom) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 4194304
                _packetBuffer[12+_cBBO+4+4+1] |= 64;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+54] = highByte(bottom);
                _packetBuffer[12+_cBBO+4+4+55] = lowByte(bottom);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Left
             * mE    0: ME1, 1: ME2
             * keyer     0-3: Keyer 1-4
             * left      -16000-16000: -9.00-9.00
             */
            void ATEMext::setKeyDVELeft(uint8_t mE, uint8_t keyer, int16_
t left) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 8388608
                _packetBuffer[12+_cBBO+4+4+1] |= 128;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+56] = highByte(left);
                _packetBuffer[12+_cBBO+4+4+57] = lowByte(left);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Right
             * mE    0: ME1, 1: ME2
```

```
             * keyer    0-3: Keyer 1-4
             * right    -16000-16000: -9.00-9.00
             */
            void ATEMext::setKeyDVERight(uint8_t mE, uint8_t keyer, int16
_t right) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 16777216
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+58] = highByte(right);
                _packetBuffer[12+_cBBO+4+4+59] = lowByte(right);

                _finishCommandPacket();

            }

            /**
             * Set Key DVE; Rate
             * mE   0: ME1, 1: ME2
             * keyer    0-3: Keyer 1-4
             * rate     1-250: Frames
             */
            void ATEMext::setKeyDVERate(uint8_t mE, uint8_t keyer, uint8_
t rate) {

                _prepareCommandPacket(("CKDV"),64,(_packetBuffer[12+_cBBO
+4+4+4]==mE) && (_packetBuffer[12+_cBBO+4+4+5]==keyer));

                    // Set Mask: 33554432
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+4] = mE;

                _packetBuffer[12+_cBBO+4+4+5] = keyer;

                _packetBuffer[12+_cBBO+4+4+60] = rate;

                _finishCommandPacket();

            }

            /**
```

```
     * Set Keyer Fly; Key Frame
     * mE   0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     * keyFrame     1: A, 2: B
     */
    void ATEMext::setKeyerFlyKeyFrame(uint8_t mE, uint8_t keyer,
uint8_t keyFrame) {

        _prepareCommandPacket(("SFKF"),4,(_packetBuffer[12+_cBBO+
4+4+0]==mE) && (_packetBuffer[12+_cBBO+4+4+1]==keyer));

        _packetBuffer[12+_cBBO+4+4+0] = mE;

        _packetBuffer[12+_cBBO+4+4+1] = keyer;

        _packetBuffer[12+_cBBO+4+4+2] = keyFrame;

        _finishCommandPacket();

    }

    /**
     * Set Run Flying Key; Key Frame
     * mE   0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     * keyFrame     1: A, 2: B, 3: Full, 4: Run-To-Infinite
     */
    void ATEMext::setRunFlyingKeyKeyFrame(uint8_t mE, uint8_t key
er, uint8_t keyFrame) {

        _prepareCommandPacket(("RFlK"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

        _packetBuffer[12+_cBBO+4+4+1] = mE;

        _packetBuffer[12+_cBBO+4+4+2] = keyer;

        _packetBuffer[12+_cBBO+4+4+4] = keyFrame;

        _finishCommandPacket();

    }

    /**
     * Set Run Flying Key; Run-to-Infinite-index
     * mE   0: ME1, 1: ME2
     * keyer    0-3: Keyer 1-4
     * runtoInfiniteindex
     */
```

```
            void ATEMext::setRunFlyingKeyRuntoInfiniteindex(uint8_t mE, u
int8_t keyer, uint8_t runtoInfiniteindex) {

                _prepareCommandPacket(("RFlK"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mE) && (_packetBuffer[12+_cBBO+4+4+2]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mE;

                _packetBuffer[12+_cBBO+4+4+2] = keyer;

                _packetBuffer[12+_cBBO+4+4+5] = runtoInfiniteindex;

                _finishCommandPacket();

            }

            /**
             * Get Downstream Keyer; Fill Source
             * keyer    0: DSK1, 1: DSK2
             */
            uint16_t ATEMext::getDownstreamKeyerFillSource(uint8_t keyer)
 {
                return atemDownstreamKeyerFillSource[keyer];
            }

            /**
             * Get Downstream Keyer; Key Source
             * keyer    0: DSK1, 1: DSK2
             */
            uint16_t ATEMext::getDownstreamKeyerKeySource(uint8_t keyer)
{
                return atemDownstreamKeyerKeySource[keyer];
            }

            /**
             * Set Downstream Keyer; Fill Source
             * keyer    0-3: Keyer 1-4
             * fillSource   (See video source list)
             */
            void ATEMext::setDownstreamKeyerFillSource(uint8_t keyer, uin
t16_t fillSource) {

                _prepareCommandPacket(("CDsF"),4,(_packetBuffer[12+_cBBO+
4+4+0]==keyer));

                _packetBuffer[12+_cBBO+4+4+0] = keyer;
```

```
            _packetBuffer[12+_cBBO+4+4+2] = highByte(fillSource);
            _packetBuffer[12+_cBBO+4+4+3] = lowByte(fillSource);

            _finishCommandPacket();

        }

        /**
         * Set Downstream Keyer; Key Source
         * keyer    0-3: Keyer 1-4
         * keySource    (See video source list)
         */
        void ATEMext::setDownstreamKeyerKeySource(uint8_t keyer, uint
16_t keySource) {

            _prepareCommandPacket(("CDsC"),4,(_packetBuffer[12+_cBBO+
4+4+0]==keyer));

            _packetBuffer[12+_cBBO+4+4+0] = keyer;

            _packetBuffer[12+_cBBO+4+4+2] = highByte(keySource);
            _packetBuffer[12+_cBBO+4+4+3] = lowByte(keySource);

            _finishCommandPacket();

        }

        /**
         * Get Downstream Keyer; Tie
         * keyer    0: DSK1, 1: DSK2
         */
        bool ATEMext::getDownstreamKeyerTie(uint8_t keyer) {
            return atemDownstreamKeyerTie[keyer];
        }

        /**
         * Get Downstream Keyer; Rate
         * keyer    0: DSK1, 1: DSK2
         */
        uint8_t ATEMext::getDownstreamKeyerRate(uint8_t keyer) {
            return atemDownstreamKeyerRate[keyer];
        }

        /**
         * Get Downstream Keyer; Pre Multiplied
         * keyer    0: DSK1, 1: DSK2
         */
```

```
bool ATEMext::getDownstreamKeyerPreMultiplied(uint8_t keyer)
{
    return atemDownstreamKeyerPreMultiplied[keyer];
}

/**
 * Get Downstream Keyer; Clip
 * keyer    0: DSK1, 1: DSK2
 */
uint16_t ATEMext::getDownstreamKeyerClip(uint8_t keyer) {
    return atemDownstreamKeyerClip[keyer];
}

/**
 * Get Downstream Keyer; Gain
 * keyer    0: DSK1, 1: DSK2
 */
uint16_t ATEMext::getDownstreamKeyerGain(uint8_t keyer) {
    return atemDownstreamKeyerGain[keyer];
}

/**
 * Get Downstream Keyer; Invert Key
 * keyer    0: DSK1, 1: DSK2
 */
bool ATEMext::getDownstreamKeyerInvertKey(uint8_t keyer) {
    return atemDownstreamKeyerInvertKey[keyer];
}

/**
 * Get Downstream Keyer; Masked
 * keyer    0: DSK1, 1: DSK2
 */
bool ATEMext::getDownstreamKeyerMasked(uint8_t keyer) {
    return atemDownstreamKeyerMasked[keyer];
}

/**
 * Get Downstream Keyer; Top
 * keyer    0: DSK1, 1: DSK2
 */
int16_t ATEMext::getDownstreamKeyerTop(uint8_t keyer) {
    return atemDownstreamKeyerTop[keyer];
}

/**
 * Get Downstream Keyer; Bottom
 * keyer    0: DSK1, 1: DSK2
 */
```

```
            int16_t ATEMext::getDownstreamKeyerBottom(uint8_t keyer) {
                return atemDownstreamKeyerBottom[keyer];
            }

            /**
             * Get Downstream Keyer; Left
             * keyer    0: DSK1, 1: DSK2
             */
            int16_t ATEMext::getDownstreamKeyerLeft(uint8_t keyer) {
                return atemDownstreamKeyerLeft[keyer];
            }

            /**
             * Get Downstream Keyer; Right
             * keyer    0: DSK1, 1: DSK2
             */
            int16_t ATEMext::getDownstreamKeyerRight(uint8_t keyer) {
                return atemDownstreamKeyerRight[keyer];
            }

            /**
             * Set Downstream Keyer; Tie
             * keyer    0: DSK1, 1: DSK2
             * tie  Bit 0: On/Off
             */
            void ATEMext::setDownstreamKeyerTie(uint8_t keyer, bool tie)
{

                _prepareCommandPacket(("CDsT"),4,(_packetBuffer[12+_cBBO+
4+4+0]==keyer));

                _packetBuffer[12+_cBBO+4+4+0] = keyer;

                _packetBuffer[12+_cBBO+4+4+1] = tie;

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Rate
             * keyer    0: DSK1, 1: DSK2
             * rate     1-250: Frames
             */
            void ATEMext::setDownstreamKeyerRate(uint8_t keyer, uint8_t r
ate) {

                _prepareCommandPacket(("CDsR"),4,(_packetBuffer[12+_cBBO+
4+4+0]==keyer));
```

```
                _packetBuffer[12+_cBBO+4+4+0] = keyer;

                _packetBuffer[12+_cBBO+4+4+1] = rate;

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Pre Multiplied
             * keyer    0-3: Keyer 1-4
             * preMultiplied    Bit 0: On/Off
             */
            void ATEMext::setDownstreamKeyerPreMultiplied(uint8_t keyer,
bool preMultiplied) {

                _prepareCommandPacket(("CDsG"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+2] = preMultiplied;

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Clip
             * keyer    0-3: Keyer 1-4
             * clip     0-1000: 0-100%
             */
            void ATEMext::setDownstreamKeyerClip(uint8_t keyer, uint16_t
clip) {

                _prepareCommandPacket(("CDsG"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(clip);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(clip);
```

```
        _finishCommandPacket();

    }

    /**
     * Set Downstream Keyer; Gain
     * keyer    0-3: Keyer 1-4
     * gain     0-1000: 0-100%
     */
    void ATEMext::setDownstreamKeyerGain(uint8_t keyer, uint16_t
gain) {

        _prepareCommandPacket(("CDsG"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

            // Set Mask: 4
        _packetBuffer[12+_cBBO+4+4+0] |= 4;

        _packetBuffer[12+_cBBO+4+4+1] = keyer;

        _packetBuffer[12+_cBBO+4+4+6] = highByte(gain);
        _packetBuffer[12+_cBBO+4+4+7] = lowByte(gain);

        _finishCommandPacket();

    }

    /**
     * Set Downstream Keyer; Invert Key(??)
     * keyer    0-3: Keyer 1-4
     * invertKey    Bit 0: On/Off
     */
    void ATEMext::setDownstreamKeyerInvertKey(uint8_t keyer, bool
 invertKey) {

        _prepareCommandPacket(("CDsG"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

            // Set Mask: 8
        _packetBuffer[12+_cBBO+4+4+0] |= 8;

        _packetBuffer[12+_cBBO+4+4+1] = keyer;

        _packetBuffer[12+_cBBO+4+4+8] = invertKey;

        _finishCommandPacket();

    }
```

```
            /**
             * Set Downstream Keyer; Masked
             * keyer    0-3: Keyer 1-4
             * masked   Bit 0: On/Off
             */
            void ATEMext::setDownstreamKeyerMasked(uint8_t keyer, bool ma
sked) {

                _prepareCommandPacket(("CDsM"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+2] = masked;

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Top
             * keyer    0-3: Keyer 1-4
             * top  -9000-9000: -9.00-9.00
             */
            void ATEMext::setDownstreamKeyerTop(uint8_t keyer, int16_t to
p) {

                _prepareCommandPacket(("CDsM"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(top);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(top);

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer; Bottom
             * keyer    0-3: Keyer 1-4
```

```
 * bottom   -9000-9000: -9.00-9.00
 */
void ATEMext::setDownstreamKeyerBottom(uint8_t keyer, int16_t
 bottom) {

        _prepareCommandPacket(("CDsM"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

            // Set Mask: 4
        _packetBuffer[12+_cBBO+4+4+0] |= 4;

        _packetBuffer[12+_cBBO+4+4+1] = keyer;

        _packetBuffer[12+_cBBO+4+4+6] = highByte(bottom);
        _packetBuffer[12+_cBBO+4+4+7] = lowByte(bottom);

        _finishCommandPacket();

}

/**
 * Set Downstream Keyer; Left
 * keyer    0-3: Keyer 1-4
 * left     -16000-16000: -9.00-9.00
 */
void ATEMext::setDownstreamKeyerLeft(uint8_t keyer, int16_t l
eft) {

        _prepareCommandPacket(("CDsM"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

            // Set Mask: 8
        _packetBuffer[12+_cBBO+4+4+0] |= 8;

        _packetBuffer[12+_cBBO+4+4+1] = keyer;

        _packetBuffer[12+_cBBO+4+4+8] = highByte(left);
        _packetBuffer[12+_cBBO+4+4+9] = lowByte(left);

        _finishCommandPacket();

}

/**
 * Set Downstream Keyer; Right
 * keyer    0-3: Keyer 1-4
 * right    -16000-16000: -9.00-9.00
 */
```

```
            void ATEMext::setDownstreamKeyerRight(uint8_t keyer, int16_t
right) {

                _prepareCommandPacket(("CDsM"),12,(_packetBuffer[12+_cBBO
+4+4+1]==keyer));

                    // Set Mask: 16
                _packetBuffer[12+_cBBO+4+4+0] |= 16;

                _packetBuffer[12+_cBBO+4+4+1] = keyer;

                _packetBuffer[12+_cBBO+4+4+10] = highByte(right);
                _packetBuffer[12+_cBBO+4+4+11] = lowByte(right);

                _finishCommandPacket();

            }

            /**
             * Set Downstream Keyer Auto; Keyer
             * keyer    0: DSK1, 1: DSK2
             */
            void ATEMext::performDownstreamKeyerAutoKeyer(uint8_t keyer)
{

                _prepareCommandPacket(("DDsA"),4);

                _packetBuffer[12+_cBBO+4+4+0] = keyer;

                _finishCommandPacket();

            }

            /**
             * Get Downstream Keyer; On Air
             * keyer    0: DSK1, 1: DSK2
             */
            bool ATEMext::getDownstreamKeyerOnAir(uint8_t keyer) {
                return atemDownstreamKeyerOnAir[keyer];
            }

            /**
             * Get Downstream Keyer; In Transition
             * keyer    0: DSK1, 1: DSK2
             */
            bool ATEMext::getDownstreamKeyerInTransition(uint8_t keyer) {
                return atemDownstreamKeyerInTransition[keyer];
            }
```

```
/**
 * Get Downstream Keyer; Is Auto Transitioning
 * keyer    0: DSK1, 1: DSK2
 */
bool ATEMext::getDownstreamKeyerIsAutoTransitioning(uint8_t k
eyer) {
    return atemDownstreamKeyerIsAutoTransitioning[keyer];
}

/**
 * Get Downstream Keyer; Frames Remaining
 * keyer    0: DSK1, 1: DSK2
 */
uint8_t ATEMext::getDownstreamKeyerFramesRemaining(uint8_t ke
yer) {
    return atemDownstreamKeyerFramesRemaining[keyer];
}

/**
 * Set Downstream Keyer; On Air
 * keyer    0: DSK1, 1: DSK2
 * onAir    Bit 0: On/Off
 */
void ATEMext::setDownstreamKeyerOnAir(uint8_t keyer, bool onA
ir) {

    _prepareCommandPacket(("CDsL"),4,(_packetBuffer[12+_cBBO+
4+4+0]==keyer));

    _packetBuffer[12+_cBBO+4+4+0] = keyer;

    _packetBuffer[12+_cBBO+4+4+1] = onAir;

    _finishCommandPacket();

}

/**
 * Get Fade-To-Black; Rate
 * mE   0: ME1, 1: ME2
 */
uint8_t ATEMext::getFadeToBlackRate(uint8_t mE) {
    return atemFadeToBlackRate[mE];
}

/**
 * Set Fade-To-Black; Rate
 * mE   0: ME1, 1: ME2
 * rate     1-250: Frames
```

```
 */
void ATEMext::setFadeToBlackRate(uint8_t mE, uint8_t rate) {

    _prepareCommandPacket(("FtbC"),4,(_packetBuffer[12+_cBBO+
4+4+1]==mE));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+1] = mE;

    _packetBuffer[12+_cBBO+4+4+2] = rate;

    _finishCommandPacket();

}

/**
 * Get Fade-To-Black State; Fully Black
 * mE    0: ME1, 1: ME2
 */
bool ATEMext::getFadeToBlackStateFullyBlack(uint8_t mE) {
    return atemFadeToBlackStateFullyBlack[mE];
}

/**
 * Get Fade-To-Black State; In Transition
 * mE    0: ME1, 1: ME2
 */
bool ATEMext::getFadeToBlackStateInTransition(uint8_t mE) {
    return atemFadeToBlackStateInTransition[mE];
}

/**
 * Get Fade-To-Black State; Frames Remaining
 * mE    0: ME1, 1: ME2
 */
uint8_t ATEMext::getFadeToBlackStateFramesRemaining(uint8_t m
E) {
    return atemFadeToBlackStateFramesRemaining[mE];
}



    /**
     * Set Fade-To-Black; M/E
     * mE    0: ME1, 1: ME2
     */
    void ATEMext::performFadeToBlackME(uint8_t mE) {
```

```
            _prepareCommandPacket(("FtbA"),4);

            _packetBuffer[12+_cBBO+4+4+0] = mE;
            _packetBuffer[12+_cBBO+4+4+1] = 0x02;

            _finishCommandPacket();

        }

    /**
     * Get Color Generator; Hue
     * colorGenerator   0: Color Generator 1, 1: Color Generator
2
     */
    uint16_t ATEMext::getColorGeneratorHue(uint8_t colorGenerator
) {
        return atemColorGeneratorHue[colorGenerator];
    }

    /**
     * Get Color Generator; Saturation
     * colorGenerator   0: Color Generator 1, 1: Color Generator
2
     */
    uint16_t ATEMext::getColorGeneratorSaturation(uint8_t colorGe
nerator) {
        return atemColorGeneratorSaturation[colorGenerator];
    }

    /**
     * Get Color Generator; Luma
     * colorGenerator   0: Color Generator 1, 1: Color Generator
2
     */
    uint16_t ATEMext::getColorGeneratorLuma(uint8_t colorGenerato
r) {
        return atemColorGeneratorLuma[colorGenerator];
    }

    /**
     * Set Color Generator; Hue
     * colorGenerator   0: Color Generator 1, 1: Color Generator
2
     * hue  0-3599: 0-359.9 Degrees
     */
    void ATEMext::setColorGeneratorHue(uint8_t colorGenerator, ui
nt16_t hue) {
```

```
                _prepareCommandPacket(("CClV"),8,(_packetBuffer[12+_cBBO+
4+4+1]==colorGenerator));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = colorGenerator;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(hue);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(hue);

                _finishCommandPacket();

            }

            /**
             * Set Color Generator; Saturation
             * colorGenerator   0: Color Generator 1, 1: Color Generator
2
             * saturation   0-1000: 0-100.0%
             */
            void ATEMext::setColorGeneratorSaturation(uint8_t colorGenera
tor, uint16_t saturation) {

                _prepareCommandPacket(("CClV"),8,(_packetBuffer[12+_cBBO+
4+4+1]==colorGenerator));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = colorGenerator;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(saturation);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(saturation);

                _finishCommandPacket();

            }

            /**
             * Set Color Generator; Luma
             * colorGenerator   0: Color Generator 1, 1: Color Generator
2
             * luma     0-1000: 0-100.0%
             */
            void ATEMext::setColorGeneratorLuma(uint8_t colorGenerator, u
int16_t luma) {
```

```
                _prepareCommandPacket(("CClV"),8,(_packetBuffer[12+_cBBO+
4+4+1]==colorGenerator));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+0] |= 4;

                _packetBuffer[12+_cBBO+4+4+1] = colorGenerator;

                _packetBuffer[12+_cBBO+4+4+6] = highByte(luma);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(luma);

                _finishCommandPacket();

            }

            /**
             * Get Aux Source; Input
             * aUXChannel    0-5: Aux 1-6
             */
            uint16_t ATEMext::getAuxSourceInput(uint8_t aUXChannel) {
                return atemAuxSourceInput[aUXChannel];
            }

            /**
             * Set Aux Source; Input
             * aUXChannel    0-5: Aux 1-6
             * input     (See video source list)
             */
            void ATEMext::setAuxSourceInput(uint8_t aUXChannel, uint16_t
input) {

                _prepareCommandPacket(("CAuS"),4,(_packetBuffer[12+_cBBO+
4+4+1]==aUXChannel));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = aUXChannel;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(input);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(input);

                _finishCommandPacket();

            }

            /**
             * Get Camera Control; Iris
             * input    1-20: Camera
```

```
 */
int16_t ATEMext::getCameraControlIris(uint8_t input) {
    return atemCameraControlIris[input];
}

/**
 * Get Camera Control; Focus
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlFocus(uint8_t input) {
    return atemCameraControlFocus[input];
}

/**
 * Get Camera Control; Gain
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlGain(uint8_t input) {
    return atemCameraControlGain[input];
}

/**
 * Get Camera Control; White Balance
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlWhiteBalance(uint8_t input)
{
    return atemCameraControlWhiteBalance[input];
}

/**
 * Get Camera Control; Sharpening Level
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlSharpeningLevel(uint8_t inpu
t) {
    return atemCameraControlSharpeningLevel[input];
}

/**
 * Get Camera Control; Zoom Normalized
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlZoomNormalized(uint8_t input
) {
    return atemCameraControlZoomNormalized[input];
}

/**
```

```
 * Get Camera Control; Zoom Speed
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlZoomSpeed(uint8_t input) {
    return atemCameraControlZoomSpeed[input];
}

/**
 * Get Camera Control; Colorbars
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlColorbars(uint8_t input) {
    return atemCameraControlColorbars[input];
}

/**
 * Get Camera Control; Lift R
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlLiftR(uint8_t input) {
    return atemCameraControlLiftR[input];
}

/**
 * Get Camera Control; Gamma R
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlGammaR(uint8_t input) {
    return atemCameraControlGammaR[input];
}

/**
 * Get Camera Control; Gain R
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlGainR(uint8_t input) {
    return atemCameraControlGainR[input];
}

/**
 * Get Camera Control; Lum Mix
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlLumMix(uint8_t input) {
    return atemCameraControlLumMix[input];
}

/**
 * Get Camera Control; Hue
```

```
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlHue(uint8_t input) {
    return atemCameraControlHue[input];
}

/**
 * Get Camera Control; Shutter
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlShutter(uint8_t input) {
    return atemCameraControlShutter[input];
}

/**
 * Get Camera Control; Lift G
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlLiftG(uint8_t input) {
    return atemCameraControlLiftG[input];
}

/**
 * Get Camera Control; Gamma G
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlGammaG(uint8_t input) {
    return atemCameraControlGammaG[input];
}

/**
 * Get Camera Control; Gain G
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlGainG(uint8_t input) {
    return atemCameraControlGainG[input];
}

/**
 * Get Camera Control; Contrast
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlContrast(uint8_t input) {
    return atemCameraControlContrast[input];
}

/**
 * Get Camera Control; Saturation
 * input    1-20: Camera
```

```
 */
int16_t ATEMext::getCameraControlSaturation(uint8_t input) {
    return atemCameraControlSaturation[input];
}

/**
 * Get Camera Control; Lift B
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlLiftB(uint8_t input) {
    return atemCameraControlLiftB[input];
}

/**
 * Get Camera Control; Gamma B
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlGammaB(uint8_t input) {
    return atemCameraControlGammaB[input];
}

/**
 * Get Camera Control; Gain B
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlGainB(uint8_t input) {
    return atemCameraControlGainB[input];
}

/**
 * Get Camera Control; Lift Y
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlLiftY(uint8_t input) {
    return atemCameraControlLiftY[input];
}

/**
 * Get Camera Control; Gamma Y
 * input    1-20: Camera
 */
int16_t ATEMext::getCameraControlGammaY(uint8_t input) {
    return atemCameraControlGammaY[input];
}

/**
 * Get Camera Control; Gain Y
 * input    1-20: Camera
 */
```

```
		int16_t ATEMext::getCameraControlGainY(uint8_t input) {
			return atemCameraControlGainY[input];
		}


		/**
			* Set Camera Control; Auto iris
			* Command takes no input
			*/

		void ATEMext::setCameraControlAutoIris(uint8_t input, int16_t
 autoiris) {
				_prepareCommandPacket(("CCmd"), 24);

				_packetBuffer[12+_cBBO+4+4+0] = input;

				_packetBuffer[12+_cBBO+4+4+1] = 0;
				_packetBuffer[12+_cBBO+4+4+2] = 5;

				_packetBuffer[12+_cBBO+4+4+4] = 0x00; // Data type: v
oid

				_finishCommandPacket();
		}


		/**
			* Set Camera Control; Detail level
			* 0: Off, 1: Low, 2: Medium, 3: High
			*/

		void ATEMext::setCameraControlSharpeningLevel(uint8_t input,
int16_t detail) {
				_prepareCommandPacket(("CCmd"), 20);

				_packetBuffer[12+_cBBO+4+4+0] = input;

				_packetBuffer[12+_cBBO+4+4+1] = 1;
				_packetBuffer[12+_cBBO+4+4+2] = 8;

				_packetBuffer[12+_cBBO+4+4+4] = 0x01; // Data type: i
nt8

				_packetBuffer[12+_cBBO+4+4+7] = 0x01;

				_packetBuffer[12+_cBBO+4+4+16] = detail & 0xFF;

				_finishCommandPacket();
		}
```

```
            /**
                * Set Camera Control; Auto focus
                * Command takes no input
                */

            void ATEMext::setCameraControlAutoFocus(uint8_t input, int16_
t autoiris) {
                    _prepareCommandPacket(("CCmd"), 24);

                    _packetBuffer[12+_cBBO+4+4+0] = input;

                    _packetBuffer[12+_cBBO+4+4+1] = 0;
                    _packetBuffer[12+_cBBO+4+4+2] = 1;

                    _packetBuffer[12+_cBBO+4+4+4] = 0x00; // Data type: v
oid

                    _finishCommandPacket();
            }

            /**
                * Set Camera Control; Reset all
                * Command takes no input
                */

            void ATEMext::setCameraControlResetAll(uint8_t input, int16_t
 reset) {
                    _prepareCommandPacket(("CCmd"), 24);

                    _packetBuffer[12+_cBBO+4+4+0] = input;

                    _packetBuffer[12+_cBBO+4+4+1] = 8;
                    _packetBuffer[12+_cBBO+4+4+2] = 7;

                    _packetBuffer[12+_cBBO+4+4+4] = 0x00; // Data type: v
oid

                    _finishCommandPacket();

                    // Update local state variables to reflect reset valu
es
                    atemCameraControlGammaY[input] = 0;
                    atemCameraControlGammaR[input] = 0;
                    atemCameraControlGammaG[input] = 0;
                    atemCameraControlGammaB[input] = 0;

                    atemCameraControlLiftY[input] = 0;
                    atemCameraControlLiftR[input] = 0;
                    atemCameraControlLiftG[input] = 0;
```

```
				atemCameraControlLiftB[input] = 0;

				atemCameraControlGainY[input] = 2048;
				atemCameraControlGainR[input] = 2048;
				atemCameraControlGainG[input] = 2048;
				atemCameraControlGainB[input] = 2048;

				atemCameraControlContrast[input] = 2048;
				atemCameraControlHue[input] = 0;
				atemCameraControlSaturation[input] = 2048;
		}

		/**
		 * Set Camera Control; Iris
		 * input	0-7: Camera
		 * iris	0-2048
		 */
		void ATEMext::setCameraControlIris(uint8_t input, int16_t iri
s) {

			_prepareCommandPacket(("CCmd"),24);

				// Preset values:
			_packetBuffer[12+_cBBO+4+4+1] = 0;
			_packetBuffer[12+_cBBO+4+4+2] = 3;

			_packetBuffer[12+_cBBO+4+4+4] = 0x80;	// Data type: 5.1
1 floating point
			_packetBuffer[12+_cBBO+4+4+9] = 0x01;	// One byte

			_packetBuffer[12+_cBBO+4+4+0] = input;

			_packetBuffer[12+_cBBO+4+4+16] = highByte(iris);
			_packetBuffer[12+_cBBO+4+4+17] = lowByte(iris);

			_finishCommandPacket();
		}

		/**
		 * Set Camera Control; Colorbars
		 * input	0-7: Camera
		 * colorbars: duration in secs (0=disable)
		 */
		void ATEMext::setCameraControlColorbars(uint8_t input, int16_
t colorbars) {

			_prepareCommandPacket(("CCmd"), 20);

			_packetBuffer[12+_cBBO+4+4+0] = input;
```

```
                // Preset values:
            _packetBuffer[12+_cBBO+4+4+1] = 4;
            _packetBuffer[12+_cBBO+4+4+2] = 4;

            _packetBuffer[12+_cBBO+4+4+4] = 0x01;   // Data type: int
8
            _packetBuffer[12+_cBBO+4+4+7] = 0x01;   // ?


            _packetBuffer[12+_cBBO+4+4+16] = (colorbars & 0xFF);

            _finishCommandPacket();

        }

        /**
         * Set Camera Control; Focus
         * input    0-7: Camera
         * focus    0-65535
         */
        void ATEMext::setCameraControlFocus(uint8_t input, int16_t fo
cus) {

            _prepareCommandPacket(("CCmd"),24);

                // Preset values:
            _packetBuffer[12+_cBBO+4+4+1] = 0;
            _packetBuffer[12+_cBBO+4+4+2] = 0;

            _packetBuffer[12+_cBBO+4+4+3] = 0x01;   // Relative setti
ng
            _packetBuffer[12+_cBBO+4+4+4] = 0x80;   // Data type: 5.1
1 floating point
            _packetBuffer[12+_cBBO+4+4+9] = 0x01;   // One byte

            _packetBuffer[12+_cBBO+4+4+0] = input;

            _packetBuffer[12+_cBBO+4+4+16] = highByte(focus);       /
/ Relative values...?
            _packetBuffer[12+_cBBO+4+4+17] = lowByte(focus);

            _finishCommandPacket();

        }

        /**
         * Set Camera Control; Gain
         * input    0-7: Camera
```

```
 * gain     512: 0db, 1024: 6db, 2048: 12db, 4096: 18db
 */
void ATEMext::setCameraControlGain(uint8_t input, int16_t gai
n) {

    _prepareCommandPacket(("CCmd"),24);

        // Preset values:
    _packetBuffer[12+_cBBO+4+4+1] = 1;
    _packetBuffer[12+_cBBO+4+4+2] = 1;

    _packetBuffer[12+_cBBO+4+4+4] = 0x01;
    _packetBuffer[12+_cBBO+4+4+7] = 0x01;

    _packetBuffer[12+_cBBO+4+4+0] = input;

    _packetBuffer[12+_cBBO+4+4+16] = highByte(gain);
    _packetBuffer[12+_cBBO+4+4+17] = lowByte(gain);

    _finishCommandPacket();

}

/**
 * Set Camera Control; White Balance
 * input    0-7: Camera
 *
 */
void ATEMext::setCameraControlWhiteBalance(uint8_t input, int
16_t whiteBalance) {

    _prepareCommandPacket(("CCmd"),24);

        // Preset values:
    _packetBuffer[12+_cBBO+4+4+1] = 1;
    _packetBuffer[12+_cBBO+4+4+2] = 2;

    _packetBuffer[12+_cBBO+4+4+4] = 0x02;
    _packetBuffer[12+_cBBO+4+4+9] = 0x01;

    _packetBuffer[12+_cBBO+4+4+0] = input;

    _packetBuffer[12+_cBBO+4+4+16] = highByte(whiteBalance);
    _packetBuffer[12+_cBBO+4+4+17] = lowByte(whiteBalance);

    _finishCommandPacket();

}
```

```
/**
 * Set Camera Control; Zoom Normalized
 * input    0-7: Camera
 *
 */
void ATEMext::setCameraControlZoomNormalized(uint8_t input, i
nt16_t zoomNormalized) {

    _prepareCommandPacket(("CCmd"),24);

        // Preset values:
    _packetBuffer[12+_cBBO+4+4+1] = 0;
    _packetBuffer[12+_cBBO+4+4+2] = 8;

    _packetBuffer[12+_cBBO+4+4+4] = 0x80;
    _packetBuffer[12+_cBBO+4+4+9] = 0x01;

    _packetBuffer[12+_cBBO+4+4+0] = input;

    _packetBuffer[12+_cBBO+4+4+16] = highByte(zoomNormalized)
;
    _packetBuffer[12+_cBBO+4+4+17] = lowByte(zoomNormalized);

    _finishCommandPacket();

}

/**
 * Set Camera Control; Zoom
 * input    0-7: Camera
 *
 */
void ATEMext::setCameraControlZoomSpeed(uint8_t input, int16_
t zoomSpeed) {

    _prepareCommandPacket(("CCmd"),24);

        // Preset values:
    _packetBuffer[12+_cBBO+4+4+1] = 0;
    _packetBuffer[12+_cBBO+4+4+2] = 9;

    _packetBuffer[12+_cBBO+4+4+4] = 0x80;
    _packetBuffer[12+_cBBO+4+4+9] = 0x01;

    _packetBuffer[12+_cBBO+4+4+0] = input;

    _packetBuffer[12+_cBBO+4+4+16] = highByte(zoomSpeed);
    _packetBuffer[12+_cBBO+4+4+17] = lowByte(zoomSpeed);
```

```
        _finishCommandPacket();

    }

    /**
     * Set Camera Control; Lift R
     * input    0-7: Camera
     * liftR    -4096-4096: -1.00-1.00
     */
    void ATEMext::setCameraControlLiftR(uint8_t input, int16_t li
ftR) {

        _prepareCommandPacket(("CCmd"),24);

            // Preset values:
        _packetBuffer[12+_cBBO+4+4+1] = 8;
        _packetBuffer[12+_cBBO+4+4+2] = 0;

        _packetBuffer[12+_cBBO+4+4+4] = 0x80;
        _packetBuffer[12+_cBBO+4+4+9] = 0x04;

        _packetBuffer[12+_cBBO+4+4+0] = input;

        _packetBuffer[12+_cBBO+4+4+16] = highByte(liftR);
        _packetBuffer[12+_cBBO+4+4+17] = lowByte(liftR);

        _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lLiftG(input));
        _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
LiftG(input));
        _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lLiftB(input));
        _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
LiftB(input));
        _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lLiftY(input));
        _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
LiftY(input));

        _finishCommandPacket();

    }

    /**
     * Set Camera Control; Gamma R
     * input    0-7: Camera
     * gammaR   -4096-4096: -1.00-1.00
     */
```

```
        void ATEMext::setCameraControlGammaR(uint8_t input, int16_t g
ammaR) {

            _prepareCommandPacket(("CCmd"),24);

                // Preset values:
            _packetBuffer[12+_cBBO+4+4+1] = 8;
            _packetBuffer[12+_cBBO+4+4+2] = 1;

            _packetBuffer[12+_cBBO+4+4+4] = 0x80;
            _packetBuffer[12+_cBBO+4+4+9] = 0x04;

            _packetBuffer[12+_cBBO+4+4+0] = input;

            _packetBuffer[12+_cBBO+4+4+16] = highByte(gammaR);
            _packetBuffer[12+_cBBO+4+4+17] = lowByte(gammaR);

            _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lGammaG(input));
            _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
GammaG(input));
            _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lGammaB(input));
            _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
GammaB(input));
            _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lGammaY(input));
            _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
GammaY(input));


            _finishCommandPacket();
        }

        /**
         * Set Camera Control; Gain R
         * input    0-7: Camera
         * gainR    -4096-4096: -1.00-1.00
         */
        void ATEMext::setCameraControlGainR(uint8_t input, int16_t ga
inR) {

            _prepareCommandPacket(("CCmd"),24);

                // Preset values:
            _packetBuffer[12+_cBBO+4+4+1] = 8;
            _packetBuffer[12+_cBBO+4+4+2] = 2;

            _packetBuffer[12+_cBBO+4+4+4] = 0x80;
```

```
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(gainR);
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(gainR);

                _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lGainG(input));
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
GainG(input));
                _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lGainB(input));
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
GainB(input));
                _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lGainY(input));
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
GainY(input));

                _finishCommandPacket();
            }

            /**
             * Set Camera Control; Lum Mix
             * input    0-7: Camera
             * lumMix   0-2048: 0-100%
             */
            void ATEMext::setCameraControlLumMix(uint8_t input, int16_t l
umMix) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 5;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x01;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(lumMix);
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(lumMix);

                _finishCommandPacket();

            }
```

```
		/**
		 * Set Camera Control; Hue
		 * input    0-7: Camera
		 * hue  -2048-2048: 0-360 degrees
		 */
		void ATEMext::setCameraControlHue(uint8_t input, int16_t hue)
 {

			_prepareCommandPacket(("CCmd"),24);

				// Preset values:
			_packetBuffer[12+_cBBO+4+4+1] = 8;
			_packetBuffer[12+_cBBO+4+4+2] = 6;

			_packetBuffer[12+_cBBO+4+4+4] = 0x80;
			_packetBuffer[12+_cBBO+4+4+9] = 0x02;

			_packetBuffer[12+_cBBO+4+4+0] = input;

			_packetBuffer[12+_cBBO+4+4+16] = highByte(hue);
			_packetBuffer[12+_cBBO+4+4+17] = lowByte(hue);

			_packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lSaturation(input));
			_packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
Saturation(input));

			_finishCommandPacket();

		}

		/**
		 * Set Camera Control; Shutter
		 * input    0-7: Camera
		 * shutter  20000: 1/50, 16667: 1/60, 13333: 1/75, 11111: 1/9
0, 10000: 1/100, 8333: 1/120, 6667: 1/150, 5556: 1/180, 4000: 1/250, 2778
: 1/360, 2000: 1/500, 1379: 1/750, 1000: 1/1000, 690: 1/1450, 500: 1/2000
		 */
		void ATEMext::setCameraControlShutter(uint8_t input, int16_t
shutter) {

			_prepareCommandPacket(("CCmd"),24);

				// Preset values:
			_packetBuffer[12+_cBBO+4+4+1] = 1;
			_packetBuffer[12+_cBBO+4+4+2] = 5;

			_packetBuffer[12+_cBBO+4+4+4] = 0x03;
			_packetBuffer[12+_cBBO+4+4+11] = 0x01;
```

```
                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+18] = highByte(shutter);
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(shutter);

                _finishCommandPacket();

            }

            /**
             * Set Camera Control; Lift G
             * input    0-7: Camera
             * liftG    -4096-4096: -1.00-1.00
             */
            void ATEMext::setCameraControlLiftG(uint8_t input, int16_t li
ftG) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 0;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lLiftR(input));
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
LiftR(input));

                _packetBuffer[12+_cBBO+4+4+18] = highByte(liftG);
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(liftG);

                _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lLiftB(input));
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
LiftB(input));
                _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lLiftY(input));
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
LiftY(input));


                _finishCommandPacket();
```

```
            }

            /**
             * Set Camera Control; Gamma G
             * input    0-7: Camera
             * gammaG   -4096-4096: -1.00-1.00
             */
            void ATEMext::setCameraControlGammaG(uint8_t input, int16_t g
ammaG) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 1;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lGammaR(input));
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
GammaR(input));

                _packetBuffer[12+_cBBO+4+4+18] = highByte(gammaG);
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(gammaG);

                _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lGammaB(input));
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
GammaB(input));
                _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lGammaY(input));
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
GammaY(input));

                _finishCommandPacket();

            }

            /**
             * Set Camera Control; Gain G
             * input    0-7: Camera
             * gainG    -4096-4096: -1.00-1.00
             */
            void ATEMext::setCameraControlGainG(uint8_t input, int16_t ga
inG) {
```

```
                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 2;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lGainR(input));
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
GainR(input));

                _packetBuffer[12+_cBBO+4+4+18] = highByte(gainG);
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(gainG);

                _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lGainB(input));
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
GainB(input));
                _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lGainY(input));
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
GainY(input));

                _finishCommandPacket();

            }

            /**
             * Set Camera Control; Contrast
             * input    0-7: Camera
             * contrast     0-4096: 0-100%
             */
            void ATEMext::setCameraControlContrast(uint8_t input, int16_t
 contrast) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 4;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x02;
```

```
        _packetBuffer[12+_cBBO+4+4+0] = input;

        // Pivot = 0.5 (Fixed16 1024)
        _packetBuffer[12+_cBBO+4+4+16] = 4;
        _packetBuffer[12+_cBBO+4+4+17] = 0;

        _packetBuffer[12+_cBBO+4+4+18] = highByte(contrast);
        _packetBuffer[12+_cBBO+4+4+19] = lowByte(contrast);

        _finishCommandPacket();

    }

    /**
     * Set Camera Control; Saturation
     * input    0-7: Camera
     * saturation   0-4096: 0-100%
     */
    void ATEMext::setCameraControlSaturation(uint8_t input, int16
_t saturation) {

        _prepareCommandPacket(("CCmd"),24);

            // Preset values:
        _packetBuffer[12+_cBBO+4+4+1] = 8;
        _packetBuffer[12+_cBBO+4+4+2] = 6;

        _packetBuffer[12+_cBBO+4+4+4] = 0x80;
        _packetBuffer[12+_cBBO+4+4+9] = 0x02;

        _packetBuffer[12+_cBBO+4+4+0] = input;

        _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lHue(input));
        _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
Hue(input));

        _packetBuffer[12+_cBBO+4+4+18] = highByte(saturation);
        _packetBuffer[12+_cBBO+4+4+19] = lowByte(saturation);

        _finishCommandPacket();

    }

    /**
     * Set Camera Control; Lift B
     * input    0-7: Camera
     * liftB    -4096-4096: -1.00-1.00
```

```
             */
            void ATEMext::setCameraControlLiftB(uint8_t input, int16_t li
ftB) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 0;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lLiftR(input));
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
LiftR(input));
                _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lLiftG(input));
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
LiftG(input));

                _packetBuffer[12+_cBBO+4+4+20] = highByte(liftB);
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(liftB);

                _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lLiftY(input));
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
LiftY(input));

                _finishCommandPacket();

            }

            /**
             * Set Camera Control; Gamma B
             * input    0-7: Camera
             * gammaB   -4096-4096: -1.00-1.00
             */
            void ATEMext::setCameraControlGammaB(uint8_t input, int16_t g
ammaB) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 1;
```

```
                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lGammaR(input));
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
GammaR(input));
                _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lGammaG(input));
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
GammaG(input));

                _packetBuffer[12+_cBBO+4+4+20] = highByte(gammaB);
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(gammaB);

                _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lGammaY(input));
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
GammaY(input));

                _finishCommandPacket();

            }

            /**
             * Set Camera Control; Gain B
             * input    0-7: Camera
             * gainB    -4096-4096: -1.00-1.00
             */
            void ATEMext::setCameraControlGainB(uint8_t input, int16_t ga
inB) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 2;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lGainR(input));
```

```
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
GainR(input));
                _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lGainG(input));
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
GainG(input));

                _packetBuffer[12+_cBBO+4+4+20] = highByte(gainB);
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(gainB);

                _packetBuffer[12+_cBBO+4+4+22] = highByte(getCameraContro
lGainY(input));
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(getCameraControl
GainY(input));

                _finishCommandPacket();

            }

            /**
             * Set Camera Control; Lift Y
             * input    0-7: Camera
             * liftY    -4096-4096: -1.00-1.00
             */
            void ATEMext::setCameraControlLiftY(uint8_t input, int16_t li
ftY) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 0;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lLiftR(input));
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
LiftR(input));
                _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lLiftG(input));
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
LiftG(input));
                _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lLiftB(input));
```

```
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
LiftB(input));

                _packetBuffer[12+_cBBO+4+4+22] = highByte(liftY);
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(liftY);

                _finishCommandPacket();

            }

            /**
             * Set Camera Control; Gamma Y
             * input    0-7: Camera
             * gammaY   -4096-4096: -1.00-1.00
             */
            void ATEMext::setCameraControlGammaY(uint8_t input, int16_t g
ammaY) {

                _prepareCommandPacket(("CCmd"),24);

                    // Preset values:
                _packetBuffer[12+_cBBO+4+4+1] = 8;
                _packetBuffer[12+_cBBO+4+4+2] = 1;

                _packetBuffer[12+_cBBO+4+4+4] = 0x80;
                _packetBuffer[12+_cBBO+4+4+9] = 0x04;

                _packetBuffer[12+_cBBO+4+4+0] = input;

                _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lGammaR(input));
                _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
GammaR(input));
                _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lGammaG(input));
                _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
GammaG(input));
                _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lGammaB(input));
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
GammaB(input));

                _packetBuffer[12+_cBBO+4+4+22] = highByte(gammaY);
                _packetBuffer[12+_cBBO+4+4+23] = lowByte(gammaY);


                _finishCommandPacket();

            }
```

```
        /**
         * Set Camera Control; Gain Y
         * input    0-7: Camera
         * gainY    -4096-4096: -1.00-1.00
         */
        void ATEMext::setCameraControlGainY(uint8_t input, int16_t ga
inY) {

            _prepareCommandPacket(("CCmd"),24);

                // Preset values:
            _packetBuffer[12+_cBBO+4+4+1] = 8;
            _packetBuffer[12+_cBBO+4+4+2] = 2;

            _packetBuffer[12+_cBBO+4+4+4] = 0x80;
            _packetBuffer[12+_cBBO+4+4+9] = 0x04;

            _packetBuffer[12+_cBBO+4+4+0] = input;

            _packetBuffer[12+_cBBO+4+4+16] = highByte(getCameraContro
lGainR(input));
            _packetBuffer[12+_cBBO+4+4+17] = lowByte(getCameraControl
GainR(input));
            _packetBuffer[12+_cBBO+4+4+18] = highByte(getCameraContro
lGainG(input));
            _packetBuffer[12+_cBBO+4+4+19] = lowByte(getCameraControl
GainG(input));
            _packetBuffer[12+_cBBO+4+4+20] = highByte(getCameraContro
lGainB(input));
            _packetBuffer[12+_cBBO+4+4+21] = lowByte(getCameraControl
GainB(input));

            _packetBuffer[12+_cBBO+4+4+22] = highByte(gainY);
            _packetBuffer[12+_cBBO+4+4+23] = lowByte(gainY);


            _finishCommandPacket();

        }

        /**
         * Get Clip Player; Playing
         * mediaPlayer  0: Media Player 1, 1: Media Player 2
         */
        bool ATEMext::getClipPlayerPlaying(uint8_t mediaPlayer) {
            return atemClipPlayerPlaying[mediaPlayer];
        }
```

```
/**
 * Get Clip Player; Loop
 * mediaPlayer  0: Media Player 1, 1: Media Player 2
 */
bool ATEMext::getClipPlayerLoop(uint8_t mediaPlayer) {
    return atemClipPlayerLoop[mediaPlayer];
}

/**
 * Get Clip Player; At Beginning
 * mediaPlayer  0: Media Player 1, 1: Media Player 2
 */
bool ATEMext::getClipPlayerAtBeginning(uint8_t mediaPlayer) {
    return atemClipPlayerAtBeginning[mediaPlayer];
}

/**
 * Get Clip Player; Clip Frame
 * mediaPlayer  0: Media Player 1, 1: Media Player 2
 */
uint16_t ATEMext::getClipPlayerClipFrame(uint8_t mediaPlayer)
 {
    return atemClipPlayerClipFrame[mediaPlayer];
}

/**
 * Set Clip Player; Playing
 * mediaPlayer  0: Media Player 1, 1: Media Player 2
 * playing  Bit 0: On/Off
 */
void ATEMext::setClipPlayerPlaying(uint8_t mediaPlayer, bool
playing) {

    _prepareCommandPacket(("SCPS"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mediaPlayer));

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

    _packetBuffer[12+_cBBO+4+4+2] = playing;

    _finishCommandPacket();

}

/**
 * Set Clip Player; Loop
```

```
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             * loop      Bit 0: On/Off
             */
            void ATEMext::setClipPlayerLoop(uint8_t mediaPlayer, bool loo
p) {

                _prepareCommandPacket(("SCPS"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mediaPlayer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

                _packetBuffer[12+_cBBO+4+4+3] = loop;

                _finishCommandPacket();

            }

            /**
             * Set Clip Player; At Beginning
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             * atBeginning  Bit 0: On/Off
             */
            void ATEMext::setClipPlayerAtBeginning(uint8_t mediaPlayer, b
ool atBeginning) {

                _prepareCommandPacket(("SCPS"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mediaPlayer));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+0] |= 4;

                _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

                _packetBuffer[12+_cBBO+4+4+4] = atBeginning;

                _finishCommandPacket();

            }

            /**
             * Set Clip Player; Clip Frame
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             * clipFrame
             */
            void ATEMext::setClipPlayerClipFrame(uint8_t mediaPlayer, uin
t16_t clipFrame) {
```

```
                _prepareCommandPacket(("SCPS"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mediaPlayer));

                    // Set Mask: 8
                _packetBuffer[12+_cBBO+4+4+0] |= 8;

                _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

                _packetBuffer[12+_cBBO+4+4+6] = highByte(clipFrame);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(clipFrame);

                _finishCommandPacket();

            }

            /**
             * Get Media Player Source; Type
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             */
            uint8_t ATEMext::getMediaPlayerSourceType(uint8_t mediaPlayer
) {
                return atemMediaPlayerSourceType[mediaPlayer];
            }

            /**
             * Get Media Player Source; Still Index
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             */
            uint8_t ATEMext::getMediaPlayerSourceStillIndex(uint8_t media
Player) {
                return atemMediaPlayerSourceStillIndex[mediaPlayer];
            }

            /**
             * Get Media Player Source; Clip Index
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             */
            uint8_t ATEMext::getMediaPlayerSourceClipIndex(uint8_t mediaP
layer) {
                return atemMediaPlayerSourceClipIndex[mediaPlayer];
            }

            /**
             * Set Media Player Source; Type
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             * type     1: Still, 2: Clip
             */
```

```
            void ATEMext::setMediaPlayerSourceType(uint8_t mediaPlayer, u
int8_t type) {

                _prepareCommandPacket(("MPSS"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mediaPlayer));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+0] |= 1;

                _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

                _packetBuffer[12+_cBBO+4+4+2] = type;

                _finishCommandPacket();

            }

            /**
             * Set Media Player Source; Still Index
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             * stillIndex   0-x: Still 1-x
             */
            void ATEMext::setMediaPlayerSourceStillIndex(uint8_t mediaPla
yer, uint8_t stillIndex) {

                _prepareCommandPacket(("MPSS"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mediaPlayer));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

                _packetBuffer[12+_cBBO+4+4+3] = stillIndex;

                _finishCommandPacket();

            }

            /**
             * Set Media Player Source; Clip Index
             * mediaPlayer  0: Media Player 1, 1: Media Player 2
             * clipIndex    0-x: Clip 1-x
             */
            void ATEMext::setMediaPlayerSourceClipIndex(uint8_t mediaPlay
er, uint8_t clipIndex) {

                _prepareCommandPacket(("MPSS"),8,(_packetBuffer[12+_cBBO+
4+4+1]==mediaPlayer));
```

```
            // Set Mask: 4
        _packetBuffer[12+_cBBO+4+4+0] |= 4;

        _packetBuffer[12+_cBBO+4+4+1] = mediaPlayer;

        _packetBuffer[12+_cBBO+4+4+4] = clipIndex;

        _finishCommandPacket();

    }

    /**
     * Get Media Pool Storage; Clip 1 Max Length
     */
    uint16_t ATEMext::getMediaPoolStorageClip1MaxLength() {
        return atemMediaPoolStorageClip1MaxLength;
    }

    /**
     * Get Media Pool Storage; Clip 2 Max Length
     */
    uint16_t ATEMext::getMediaPoolStorageClip2MaxLength() {
        return atemMediaPoolStorageClip2MaxLength;
    }

    /**
     * Set Media Pool Storage; Clip 1 Max Length
     * clip1MaxLength    Frames
     */
    void ATEMext::setMediaPoolStorageClip1MaxLength(uint16_t clip
1MaxLength) {

        _prepareCommandPacket(("CMPS"),4);

        _packetBuffer[12+_cBBO+4+4+0] = highByte(clip1MaxLength);
        _packetBuffer[12+_cBBO+4+4+1] = lowByte(clip1MaxLength);

        _finishCommandPacket();

    }

    /**
     * Get Media Player Clip Source; Is Used
     * clipBank     0-1: Clip Bank
     */
    bool ATEMext::getMediaPlayerClipSourceIsUsed(uint8_t clipBank
) {
        return atemMediaPlayerClipSourceIsUsed[clipBank];
```

```
}

/**
 * Get Media Player Clip Source; File Name
 * clipBank     0-1: Clip Bank
 */
char *  ATEMext::getMediaPlayerClipSourceFileName(uint8_t cli
pBank) {
    return atemMediaPlayerClipSourceFileName[clipBank];
}

/**
 * Get Media Player Clip Source; Frames
 * clipBank     0-1: Clip Bank
 */
uint16_t ATEMext::getMediaPlayerClipSourceFrames(uint8_t clip
Bank) {
    return atemMediaPlayerClipSourceFrames[clipBank];
}

/**
 * Get Media Player Audio Source; Is Used
 * clipBank     1-2: Clip Bank
 */
bool ATEMext::getMediaPlayerAudioSourceIsUsed(uint8_t clipBan
k) {
    return atemMediaPlayerAudioSourceIsUsed[clipBank];
}

/**
 * Get Media Player Audio Source; File Name
 * clipBank     1-2: Clip Bank
 */
char *  ATEMext::getMediaPlayerAudioSourceFileName(uint8_t cl
ipBank) {
    return atemMediaPlayerAudioSourceFileName[clipBank];
}

/**
 * Get Macro Run Status; State
 */
uint8_t ATEMext::getMacroRunStatusState() {
    return atemMacroRunStatusState;
}

/**
 * Get Macro Run Status; Is Looping
 */
bool ATEMext::getMacroRunStatusIsLooping() {
```

```
            return atemMacroRunStatusIsLooping;
        }

        /**
         * Get Macro Run Status; Index
         */
        uint16_t ATEMext::getMacroRunStatusIndex() {
            return atemMacroRunStatusIndex;
        }

        /**
         * Set Macro Action; Action
         * index    0-99: Macro Index Number. 0xFFFF: stop
         * action   0: Run Macro, 1: Stop (w/Index 0xFFFF), 2: Stop R
ecording (w/Index 0xFFFF), 3: Insert Wait for User (w/Index 0xFFFF), 4: C
ontinue (w/Index 0xFFFF), 5: Delete Macro
         */
        void ATEMext::setMacroAction(uint16_t index, uint8_t action)
{

            _prepareCommandPacket(("MAct"),4,(_packetBuffer[12+_cBBO+
4+4+0]==highByte(index)) && (_packetBuffer[12+_cBBO+4+4+1]==lowByte(index
)));

            _packetBuffer[12+_cBBO+4+4+0] = highByte(index);
            _packetBuffer[12+_cBBO+4+4+1] = lowByte(index);

            _packetBuffer[12+_cBBO+4+4+2] = action;

            _finishCommandPacket();

        }

        /**
         * Set Macro Run Change Properties; Looping
         * looping  Bit 0: On/Off
         */
        void ATEMext::setMacroRunChangePropertiesLooping(bool looping
) {

            _prepareCommandPacket(("MRCP"),4);

                // Set Mask: 1
            _packetBuffer[12+_cBBO+4+4+0] |= 1;

            _packetBuffer[12+_cBBO+4+4+1] = looping;

            _finishCommandPacket();
```

```
}

/**
 * Get Macro Properties; Is Used
 * macroIndex   0-9: Macro Index Number
 */
bool ATEMext::getMacroPropertiesIsUsed(uint8_t macroIndex) {
    return atemMacroPropertiesIsUsed[macroIndex];
}

/**
 * Get Macro Properties; Name
 * macroIndex   0-9: Macro Index Number
 */
char *  ATEMext::getMacroPropertiesName(uint8_t macroIndex) {
    return atemMacroPropertiesName[macroIndex];
}

/**
 * Set Macro Start Recording; Index
 * index    0-99: Macro Index Number
 */
void ATEMext::setMacroStartRecordingIndex(uint8_t index) {

    _prepareCommandPacket(("MSRc"),8);

    _packetBuffer[12+_cBBO+4+4+1] = index;

    _finishCommandPacket();

}

/**
 * Set Macro Add Pause; Frames
 * frames   Number of
 */
void ATEMext::setMacroAddPauseFrames(uint16_t frames) {

    _prepareCommandPacket(("MSlp"),4);

    _packetBuffer[12+_cBBO+4+4+2] = highByte(frames);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(frames);

    _finishCommandPacket();

}

/**
 * Get Macro Recording Status; Is Recording
```

```
 */
bool ATEMext::getMacroRecordingStatusIsRecording() {
    return atemMacroRecordingStatusIsRecording;
}

/**
 * Get Macro Recording Status; Index
 */
uint16_t ATEMext::getMacroRecordingStatusIndex() {
    return atemMacroRecordingStatusIndex;
}

/**
 * Get Super Source; Fill Source
 */
uint16_t ATEMext::getSuperSourceFillSource() {
    return atemSuperSourceFillSource;
}

/**
 * Get Super Source; Key Source
 */
uint16_t ATEMext::getSuperSourceKeySource() {
    return atemSuperSourceKeySource;
}

/**
 * Get Super Source; Foreground
 */
bool ATEMext::getSuperSourceForeground() {
    return atemSuperSourceForeground;
}

/**
 * Get Super Source; Pre Multiplied
 */
bool ATEMext::getSuperSourcePreMultiplied() {
    return atemSuperSourcePreMultiplied;
}

/**
 * Get Super Source; Clip
 */
uint16_t ATEMext::getSuperSourceClip() {
    return atemSuperSourceClip;
}

/**
 * Get Super Source; Gain
```

```
 */
uint16_t ATEMext::getSuperSourceGain() {
    return atemSuperSourceGain;
}

/**
 * Get Super Source; Invert Key
 */
bool ATEMext::getSuperSourceInvertKey() {
    return atemSuperSourceInvertKey;
}

/**
 * Get Super Source; Border Enabled
 */
bool ATEMext::getSuperSourceBorderEnabled() {
    return atemSuperSourceBorderEnabled;
}

/**
 * Get Super Source; Border Bevel
 */
uint8_t ATEMext::getSuperSourceBorderBevel() {
    return atemSuperSourceBorderBevel;
}

/**
 * Get Super Source; Border Outer Width
 */
uint16_t ATEMext::getSuperSourceBorderOuterWidth() {
    return atemSuperSourceBorderOuterWidth;
}

/**
 * Get Super Source; Border Inner Width
 */
uint16_t ATEMext::getSuperSourceBorderInnerWidth() {
    return atemSuperSourceBorderInnerWidth;
}

/**
 * Get Super Source; Border Outer Softness
 */
uint8_t ATEMext::getSuperSourceBorderOuterSoftness() {
    return atemSuperSourceBorderOuterSoftness;
}

/**
 * Get Super Source; Border Inner Softness
```

```
 */
uint8_t ATEMext::getSuperSourceBorderInnerSoftness() {
    return atemSuperSourceBorderInnerSoftness;
}

/**
 * Get Super Source; Border Bevel Softness
 */
uint8_t ATEMext::getSuperSourceBorderBevelSoftness() {
    return atemSuperSourceBorderBevelSoftness;
}

/**
 * Get Super Source; Border Bevel Position
 */
uint8_t ATEMext::getSuperSourceBorderBevelPosition() {
    return atemSuperSourceBorderBevelPosition;
}

/**
 * Get Super Source; Border Hue
 */
uint16_t ATEMext::getSuperSourceBorderHue() {
    return atemSuperSourceBorderHue;
}

/**
 * Get Super Source; Border Saturation
 */
uint16_t ATEMext::getSuperSourceBorderSaturation() {
    return atemSuperSourceBorderSaturation;
}

/**
 * Get Super Source; Border Luma
 */
uint16_t ATEMext::getSuperSourceBorderLuma() {
    return atemSuperSourceBorderLuma;
}

/**
 * Get Super Source; Light Source Direction
 */
uint16_t ATEMext::getSuperSourceLightSourceDirection() {
    return atemSuperSourceLightSourceDirection;
}

/**
 * Get Super Source; Light Source Altitude
```

```
 */
uint8_t ATEMext::getSuperSourceLightSourceAltitude() {
    return atemSuperSourceLightSourceAltitude;
}

/**
 * Set Super Source; Fill Source
 * fillSource    (See video source list)
 */
void ATEMext::setSuperSourceFillSource(uint16_t fillSource) {

    _prepareCommandPacket(("CSSc"),36);

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+3] |= 1;

    _packetBuffer[12+_cBBO+4+4+4] = highByte(fillSource);
    _packetBuffer[12+_cBBO+4+4+5] = lowByte(fillSource);

    _finishCommandPacket();

}

/**
 * Set Super Source; Key Source
 * keySource     (See video source list)
 */
void ATEMext::setSuperSourceKeySource(uint16_t keySource) {

    _prepareCommandPacket(("CSSc"),36);

        // Set Mask: 2
    _packetBuffer[12+_cBBO+4+4+3] |= 2;

    _packetBuffer[12+_cBBO+4+4+6] = highByte(keySource);
    _packetBuffer[12+_cBBO+4+4+7] = lowByte(keySource);

    _finishCommandPacket();

}

/**
 * Set Super Source; Foreground
 * foreground    Bit 0: On/Off
 */
void ATEMext::setSuperSourceForeground(bool foreground) {

    _prepareCommandPacket(("CSSc"),36);
```

```
            // Set Mask: 4
        _packetBuffer[12+_cBBO+4+4+3] |= 4;

        _packetBuffer[12+_cBBO+4+4+8] = foreground;

        _finishCommandPacket();

    }

    /**
     * Set Super Source; Pre Multiplied
     * preMultiplied    Bit 0: On/Off
     */
    void ATEMext::setSuperSourcePreMultiplied(bool preMultiplied)
{

        _prepareCommandPacket(("CSSc"),36);

            // Set Mask: 8
        _packetBuffer[12+_cBBO+4+4+3] |= 8;

        _packetBuffer[12+_cBBO+4+4+9] = preMultiplied;

        _finishCommandPacket();

    }

    /**
     * Set Super Source; Clip
     * clip     0-1000: 0-100%
     */
    void ATEMext::setSuperSourceClip(uint16_t clip) {

        _prepareCommandPacket(("CSSc"),36);

            // Set Mask: 16
        _packetBuffer[12+_cBBO+4+4+3] |= 16;

        _packetBuffer[12+_cBBO+4+4+10] = highByte(clip);
        _packetBuffer[12+_cBBO+4+4+11] = lowByte(clip);

        _finishCommandPacket();

    }

    /**
     * Set Super Source; Gain
     * gain     0-1000: 0-100%
     */
```

```
void ATEMext::setSuperSourceGain(uint16_t gain) {

    _prepareCommandPacket(("CSSc"),36);

        // Set Mask: 32
    _packetBuffer[12+_cBBO+4+4+3] |= 32;

    _packetBuffer[12+_cBBO+4+4+12] = highByte(gain);
    _packetBuffer[12+_cBBO+4+4+13] = lowByte(gain);

    _finishCommandPacket();

}

/**
 * Set Super Source; Invert Key
 * invertKey    Bit 0: On/Off
 */
void ATEMext::setSuperSourceInvertKey(bool invertKey) {

    _prepareCommandPacket(("CSSc"),36);

        // Set Mask: 64
    _packetBuffer[12+_cBBO+4+4+3] |= 64;

    _packetBuffer[12+_cBBO+4+4+14] = invertKey;

    _finishCommandPacket();

}

/**
 * Set Super Source; Border Enabled
 * borderEnabled    Bit 0: On/Off
 */
void ATEMext::setSuperSourceBorderEnabled(bool borderEnabled)
{

    _prepareCommandPacket(("CSSc"),36);

        // Set Mask: 128
    _packetBuffer[12+_cBBO+4+4+3] |= 128;

    _packetBuffer[12+_cBBO+4+4+15] = borderEnabled;

    _finishCommandPacket();

}
```

```
            /**
             * Set Super Source; Border Bevel
             * borderBevel  0: No, 1: In/Out, 2: In, 3: Out
             */
            void ATEMext::setSuperSourceBorderBevel(uint8_t borderBevel)
{

                _prepareCommandPacket(("CSSc"),36);

                    // Set Mask: 256
                _packetBuffer[12+_cBBO+4+4+2] |= 1;

                _packetBuffer[12+_cBBO+4+4+16] = borderBevel;

                _finishCommandPacket();

            }

            /**
             * Set Super Source; Border Outer Width
             * borderOuterWidth     0-1600: 0-16.00
             */
            void ATEMext::setSuperSourceBorderOuterWidth(uint16_t borderO
uterWidth) {

                _prepareCommandPacket(("CSSc"),36);

                    // Set Mask: 512
                _packetBuffer[12+_cBBO+4+4+2] |= 2;

                _packetBuffer[12+_cBBO+4+4+18] = highByte(borderOuterWidt
h);

                _packetBuffer[12+_cBBO+4+4+19] = lowByte(borderOuterWidth
);

                _finishCommandPacket();

            }

            /**
             * Set Super Source; Border Inner Width
             * borderInnerWidth     0-1600: 0-16.00
             */
            void ATEMext::setSuperSourceBorderInnerWidth(uint16_t borderI
nnerWidth) {

                _prepareCommandPacket(("CSSc"),36);

                    // Set Mask: 1024
```

```
                _packetBuffer[12+_cBBO+4+4+2] |= 4;

                _packetBuffer[12+_cBBO+4+4+20] = highByte(borderInnerWidt
h);
                _packetBuffer[12+_cBBO+4+4+21] = lowByte(borderInnerWidth
);

                _finishCommandPacket();

            }

            /**
             * Set Super Source; Border Outer Softness
             * borderOuterSoftness  0-100: 0-100%
             */
            void ATEMext::setSuperSourceBorderOuterSoftness(uint8_t borde
rOuterSoftness) {

                _prepareCommandPacket(("CSSc"),36);

                    // Set Mask: 2048
                _packetBuffer[12+_cBBO+4+4+2] |= 8;

                _packetBuffer[12+_cBBO+4+4+22] = borderOuterSoftness;

                _finishCommandPacket();

            }

            /**
             * Set Super Source; Border Inner Softness
             * borderInnerSoftness  0-100: 0-100%
             */
            void ATEMext::setSuperSourceBorderInnerSoftness(uint8_t borde
rInnerSoftness) {

                _prepareCommandPacket(("CSSc"),36);

                    // Set Mask: 4096
                _packetBuffer[12+_cBBO+4+4+2] |= 16;

                _packetBuffer[12+_cBBO+4+4+23] = borderInnerSoftness;

                _finishCommandPacket();

            }

            /**
             * Set Super Source; Border Bevel Softness
```

```
 * borderBevelSoftness  0-100: 0.0-1.0
 */
void ATEMext::setSuperSourceBorderBevelSoftness(uint8_t borde
rBevelSoftness) {

    _prepareCommandPacket(("CSSc"),36);

        // Set Mask: 8192
    _packetBuffer[12+_cBBO+4+4+2] |= 32;

    _packetBuffer[12+_cBBO+4+4+24] = borderBevelSoftness;

    _finishCommandPacket();

}

/**
 * Set Super Source; Border Bevel Position
 * borderBevelPosition  0-100: 0.0-1.0
 */
void ATEMext::setSuperSourceBorderBevelPosition(uint8_t borde
rBevelPosition) {

    _prepareCommandPacket(("CSSc"),36);

        // Set Mask: 16384
    _packetBuffer[12+_cBBO+4+4+2] |= 64;

    _packetBuffer[12+_cBBO+4+4+25] = borderBevelPosition;

    _finishCommandPacket();

}

/**
 * Set Super Source; Border Hue
 * borderHue    0-3599: 0-359.9 Degrees
 */
void ATEMext::setSuperSourceBorderHue(uint16_t borderHue) {

    _prepareCommandPacket(("CSSc"),36);

        // Set Mask: 32768
    _packetBuffer[12+_cBBO+4+4+2] |= 128;

    _packetBuffer[12+_cBBO+4+4+26] = highByte(borderHue);
    _packetBuffer[12+_cBBO+4+4+27] = lowByte(borderHue);

    _finishCommandPacket();
```

```
            }

            /**
             * Set Super Source; Border Saturation
             * borderSaturation     0-1000: 0-100%
             */
            void ATEMext::setSuperSourceBorderSaturation(uint16_t borderS
aturation) {

                _prepareCommandPacket(("CSSc"),36);

                    // Set Mask: 65536
                _packetBuffer[12+_cBBO+4+4+1] |= 1;

                _packetBuffer[12+_cBBO+4+4+28] = highByte(borderSaturatio
n);
                _packetBuffer[12+_cBBO+4+4+29] = lowByte(borderSaturation
);

                _finishCommandPacket();

            }

            /**
             * Set Super Source; Border Luma
             * borderLuma   0-1000: 0-100%
             */
            void ATEMext::setSuperSourceBorderLuma(uint16_t borderLuma) {

                _prepareCommandPacket(("CSSc"),36);

                    // Set Mask: 131072
                _packetBuffer[12+_cBBO+4+4+1] |= 2;

                _packetBuffer[12+_cBBO+4+4+30] = highByte(borderLuma);
                _packetBuffer[12+_cBBO+4+4+31] = lowByte(borderLuma);

                _finishCommandPacket();

            }

            /**
             * Set Super Source; Light Source Direction
             * lightSourceDirection     0-3590: 0-359 Degrees
             */
            void ATEMext::setSuperSourceLightSourceDirection(uint16_t lig
htSourceDirection) {
```

```
        _prepareCommandPacket(("CSSc"),36);

            // Set Mask: 262144
        _packetBuffer[12+_cBBO+4+4+1] |= 4;

        _packetBuffer[12+_cBBO+4+4+32] = highByte(lightSourceDire
ction);
        _packetBuffer[12+_cBBO+4+4+33] = lowByte(lightSourceDirec
tion);

        _finishCommandPacket();

    }

    /**
     * Set Super Source; Light Source Altitude
     * lightSourceAltitude  10-100: 10-100
     */
    void ATEMext::setSuperSourceLightSourceAltitude(uint8_t light
SourceAltitude) {

        _prepareCommandPacket(("CSSc"),36);

            // Set Mask: 524288
        _packetBuffer[12+_cBBO+4+4+1] |= 8;

        _packetBuffer[12+_cBBO+4+4+34] = lightSourceAltitude;

        _finishCommandPacket();

    }

    /**
     * Get Super Source Box Parameters; Enabled
     * box  0-3: Box 1-4
     */
    bool ATEMext::getSuperSourceBoxParametersEnabled(uint8_t box)
 {
        return atemSuperSourceBoxParametersEnabled[box];
    }

    /**
     * Get Super Source Box Parameters; Input Source
     * box  0-3: Box 1-4
     */
    uint16_t ATEMext::getSuperSourceBoxParametersInputSource(uint
8_t box) {
        return atemSuperSourceBoxParametersInputSource[box];
    }
```

```
/**
 * Get Super Source Box Parameters; Position X
 * box  0-3: Box 1-4
 */
int16_t ATEMext::getSuperSourceBoxParametersPositionX(uint8_t
box) {
    return atemSuperSourceBoxParametersPositionX[box];
}

/**
 * Get Super Source Box Parameters; Position Y
 * box  0-3: Box 1-4
 */
int16_t ATEMext::getSuperSourceBoxParametersPositionY(uint8_t
box) {
    return atemSuperSourceBoxParametersPositionY[box];
}

/**
 * Get Super Source Box Parameters; Size
 * box  0-3: Box 1-4
 */
uint16_t ATEMext::getSuperSourceBoxParametersSize(uint8_t box
) {
    return atemSuperSourceBoxParametersSize[box];
}

/**
 * Get Super Source Box Parameters; Cropped
 * box  0-3: Box 1-4
 */
bool ATEMext::getSuperSourceBoxParametersCropped(uint8_t box)
{
    return atemSuperSourceBoxParametersCropped[box];
}

/**
 * Get Super Source Box Parameters; Crop Top
 * box  0-3: Box 1-4
 */
uint16_t ATEMext::getSuperSourceBoxParametersCropTop(uint8_t
box) {
    return atemSuperSourceBoxParametersCropTop[box];
}

/**
 * Get Super Source Box Parameters; Crop Bottom
 * box  0-3: Box 1-4
```

```
             */
            uint16_t ATEMext::getSuperSourceBoxParametersCropBottom(uint8
_t box) {
                return atemSuperSourceBoxParametersCropBottom[box];
            }

            /**
             * Get Super Source Box Parameters; Crop Left
             * box  0-3: Box 1-4
             */
            uint16_t ATEMext::getSuperSourceBoxParametersCropLeft(uint8_t
 box) {
                return atemSuperSourceBoxParametersCropLeft[box];
            }

            /**
             * Get Super Source Box Parameters; Crop Right
             * box  0-3: Box 1-4
             */
            uint16_t ATEMext::getSuperSourceBoxParametersCropRight(uint8_
t box) {
                return atemSuperSourceBoxParametersCropRight[box];
            }

            /**
             * Set Super Source Box Parameters; Enabled
             * box  0-3: Box 1-4
             * enabled  Bit 0: On/Off
             */
            void ATEMext::setSuperSourceBoxParametersEnabled(uint8_t box,
 bool enabled) {

                _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                    // Set Mask: 1
                _packetBuffer[12+_cBBO+4+4+1] |= 1;

                _packetBuffer[12+_cBBO+4+4+2] = box;

                _packetBuffer[12+_cBBO+4+4+3] = enabled;

                _finishCommandPacket();

            }

            /**
             * Set Super Source Box Parameters; Input Source
             * box  0-3: Box 1-4
```

```
             * inputSource  (See video source list)
             */
            void ATEMext::setSuperSourceBoxParametersInputSource(uint8_t
box, uint16_t inputSource) {

                _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+1] |= 2;

                _packetBuffer[12+_cBBO+4+4+2] = box;

                _packetBuffer[12+_cBBO+4+4+4] = highByte(inputSource);
                _packetBuffer[12+_cBBO+4+4+5] = lowByte(inputSource);

                _finishCommandPacket();

            }

            /**
             * Set Super Source Box Parameters; Position X
             * box  0-3: Box 1-4
             * positionX    -4800-4800: -48.00-48.00
             */
            void ATEMext::setSuperSourceBoxParametersPositionX(uint8_t bo
x, int16_t positionX) {

                _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+1] |= 4;

                _packetBuffer[12+_cBBO+4+4+2] = box;

                _packetBuffer[12+_cBBO+4+4+6] = highByte(positionX);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(positionX);

                _finishCommandPacket();

            }

            /**
             * Set Super Source Box Parameters; Position Y
             * box  0-3: Box 1-4
             * positionY    -2700-2700: -27.00-27.00
             */
```

```
            void ATEMext::setSuperSourceBoxParametersPositionY(uint8_t bo
x, int16_t positionY) {

                _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                    // Set Mask: 8
                _packetBuffer[12+_cBBO+4+4+1] |= 8;

                _packetBuffer[12+_cBBO+4+4+2] = box;

                _packetBuffer[12+_cBBO+4+4+8] = highByte(positionY);
                _packetBuffer[12+_cBBO+4+4+9] = lowByte(positionY);

                _finishCommandPacket();

            }

            /**
             * Set Super Source Box Parameters; Size
             * box  0-3: Box 1-4
             * size     70-1000: 0.07-1.00
             */
            void ATEMext::setSuperSourceBoxParametersSize(uint8_t box, ui
nt16_t size) {

                _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                    // Set Mask: 16
                _packetBuffer[12+_cBBO+4+4+1] |= 16;

                _packetBuffer[12+_cBBO+4+4+2] = box;

                _packetBuffer[12+_cBBO+4+4+10] = highByte(size);
                _packetBuffer[12+_cBBO+4+4+11] = lowByte(size);

                _finishCommandPacket();

            }

            /**
             * Set Super Source Box Parameters; Cropped
             * box  0-3: Box 1-4
             * cropped  Bit 0: On/Off
             */
            void ATEMext::setSuperSourceBoxParametersCropped(uint8_t box,
 bool cropped) {
```

```
                _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                    // Set Mask: 32
                _packetBuffer[12+_cBBO+4+4+1] |= 32;

                _packetBuffer[12+_cBBO+4+4+2] = box;

                _packetBuffer[12+_cBBO+4+4+12] = cropped;

                _finishCommandPacket();

            }

            /**
             * Set Super Source Box Parameters; Crop Top
             * box  0-3: Box 1-4
             * cropTop  0-18000: 0.0-18.0
             */
            void ATEMext::setSuperSourceBoxParametersCropTop(uint8_t box,
 uint16_t cropTop) {

                _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                    // Set Mask: 64
                _packetBuffer[12+_cBBO+4+4+1] |= 64;

                _packetBuffer[12+_cBBO+4+4+2] = box;

                _packetBuffer[12+_cBBO+4+4+14] = highByte(cropTop);
                _packetBuffer[12+_cBBO+4+4+15] = lowByte(cropTop);

                _finishCommandPacket();

            }

            /**
             * Set Super Source Box Parameters; Crop Bottom
             * box  0-3: Box 1-4
             * cropBottom   0-18000: 0.0-18.0
             */
            void ATEMext::setSuperSourceBoxParametersCropBottom(uint8_t b
ox, uint16_t cropBottom) {

                _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                    // Set Mask: 128
```

```
            _packetBuffer[12+_cBBO+4+4+1] |= 128;

            _packetBuffer[12+_cBBO+4+4+2] = box;

            _packetBuffer[12+_cBBO+4+4+16] = highByte(cropBottom);
            _packetBuffer[12+_cBBO+4+4+17] = lowByte(cropBottom);

            _finishCommandPacket();

        }

        /**
         * Set Super Source Box Parameters; Crop Left
         * box  0-3: Box 1-4
         * cropLeft     0-32000: 0.0-32.0
         */
        void ATEMext::setSuperSourceBoxParametersCropLeft(uint8_t box
, uint16_t cropLeft) {

            _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                // Set Mask: 256
            _packetBuffer[12+_cBBO+4+4+0] |= 1;

            _packetBuffer[12+_cBBO+4+4+2] = box;

            _packetBuffer[12+_cBBO+4+4+18] = highByte(cropLeft);
            _packetBuffer[12+_cBBO+4+4+19] = lowByte(cropLeft);

            _finishCommandPacket();

        }

        /**
         * Set Super Source Box Parameters; Crop Right
         * box  0-3: Box 1-4
         * cropRight    0-32000: 0.0-32.0
         */
        void ATEMext::setSuperSourceBoxParametersCropRight(uint8_t bo
x, uint16_t cropRight) {

            _prepareCommandPacket(("CSBP"),24,(_packetBuffer[12+_cBBO
+4+4+2]==box));

                // Set Mask: 512
            _packetBuffer[12+_cBBO+4+4+0] |= 2;

            _packetBuffer[12+_cBBO+4+4+2] = box;
```

```
		_packetBuffer[12+_cBBO+4+4+20] = highByte(cropRight);
		_packetBuffer[12+_cBBO+4+4+21] = lowByte(cropRight);

		_finishCommandPacket();

	}

	/**
	 * Get Audio Mixer Input; Type
	 * audioSource  (See audio source list)
	 */
	uint8_t ATEMext::getAudioMixerInputType(uint16_t audioSource)
 {
		return atemAudioMixerInputType[getAudioSrcIndex(audioSour
ce)];
	}

	/**
	 * Get Audio Mixer Input; From Media Player
	 * audioSource  (See audio source list)
	 */
	bool ATEMext::getAudioMixerInputFromMediaPlayer(uint16_t audi
oSource) {
		return atemAudioMixerInputFromMediaPlayer[getAudioSrcInde
x(audioSource)];
	}

	/**
	 * Get Audio Mixer Input; Plug type
	 * audioSource  (See audio source list)
	 */
	uint8_t ATEMext::getAudioMixerInputPlugtype(uint16_t audioSou
rce) {
		return atemAudioMixerInputPlugtype[getAudioSrcIndex(audio
Source)];
	}

	/**
	 * Get Audio Mixer Input; Mix Option
	 * audioSource  (See audio source list)
	 */
	uint8_t ATEMext::getAudioMixerInputMixOption(uint16_t audioSo
urce) {
		return atemAudioMixerInputMixOption[getAudioSrcIndex(audi
oSource)];
	}

	/**
```

```
            * Get Audio Mixer Input; Volume
            * audioSource  (See audio source list)
            */
           uint16_t ATEMext::getAudioMixerInputVolume(uint16_t audioSour
ce) {
               return atemAudioMixerInputVolume[getAudioSrcIndex(audioSo
urce)];
           }

           /**
            * Get Audio Mixer Input; Balance
            * audioSource  (See audio source list)
            */
           int16_t ATEMext::getAudioMixerInputBalance(uint16_t audioSour
ce) {
               return atemAudioMixerInputBalance[getAudioSrcIndex(audioS
ource)];
           }

           /**
            * Set Audio Mixer Input; Mix Option
            * audioSource  (See audio source list)
            * mixOption    0: Off, 1: On, 2: AFV
            */
           void ATEMext::setAudioMixerInputMixOption(uint16_t audioSourc
e, uint8_t mixOption) {

               _prepareCommandPacket(("CAMI"),12,(_packetBuffer[12+_cBBO
+4+4+2]==highByte(audioSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lowByt
e(audioSource)));

                   // Set Mask: 1
               _packetBuffer[12+_cBBO+4+4+0] |= 1;

               _packetBuffer[12+_cBBO+4+4+2] = highByte(audioSource);
               _packetBuffer[12+_cBBO+4+4+3] = lowByte(audioSource);

               _packetBuffer[12+_cBBO+4+4+4] = mixOption;

               _finishCommandPacket();

           }

           /**
            * Set Audio Mixer Input; Volume
            * audioSource  (See audio source list)
            * volume   0-65381: (DB)
            */
```

```
            void ATEMext::setAudioMixerInputVolume(uint16_t audioSource,
uint16_t volume) {

                _prepareCommandPacket(("CAMI"),12,(_packetBuffer[12+_cBBO
+4+4+2]==highByte(audioSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lowByt
e(audioSource)));

                    // Set Mask: 2
                _packetBuffer[12+_cBBO+4+4+0] |= 2;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(audioSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(audioSource);

                _packetBuffer[12+_cBBO+4+4+6] = highByte(volume);
                _packetBuffer[12+_cBBO+4+4+7] = lowByte(volume);

                _finishCommandPacket();

            }

            /**
             * Set Audio Mixer Input; Balance
             * audioSource  (See audio source list)
             * balance  -10000-10000: Left/Right Extremes
             */
            void ATEMext::setAudioMixerInputBalance(uint16_t audioSource,
 int16_t balance) {

                _prepareCommandPacket(("CAMI"),12,(_packetBuffer[12+_cBBO
+4+4+2]==highByte(audioSource)) && (_packetBuffer[12+_cBBO+4+4+3]==lowByt
e(audioSource)));

                    // Set Mask: 4
                _packetBuffer[12+_cBBO+4+4+0] |= 4;

                _packetBuffer[12+_cBBO+4+4+2] = highByte(audioSource);
                _packetBuffer[12+_cBBO+4+4+3] = lowByte(audioSource);

                _packetBuffer[12+_cBBO+4+4+8] = highByte(balance);
                _packetBuffer[12+_cBBO+4+4+9] = lowByte(balance);

                _finishCommandPacket();

            }

            /**
             * Get Audio Mixer Master; Volume
             */
            uint16_t ATEMext::getAudioMixerMasterVolume() {
```

```
    return atemAudioMixerMasterVolume;
}

/**
 * Set Audio Mixer Master; Volume
 * volume   0-65381: (DB)
 */
void ATEMext::setAudioMixerMasterVolume(uint16_t volume) {

    _prepareCommandPacket(("CAMM"),8);

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(volume);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(volume);

    _finishCommandPacket();

}

/**
 * Get Audio Mixer Monitor; Monitor Audio
 */
bool ATEMext::getAudioMixerMonitorMonitorAudio() {
    return atemAudioMixerMonitorMonitorAudio;
}

/**
 * Get Audio Mixer Monitor; Volume
 */
uint16_t ATEMext::getAudioMixerMonitorVolume() {
    return atemAudioMixerMonitorVolume;
}

/**
 * Get Audio Mixer Monitor; Mute
 */
bool ATEMext::getAudioMixerMonitorMute() {
    return atemAudioMixerMonitorMute;
}

/**
 * Get Audio Mixer Monitor; Solo
 */
bool ATEMext::getAudioMixerMonitorSolo() {
    return atemAudioMixerMonitorSolo;
}
```

```
/**
 * Get Audio Mixer Monitor; Solo Input
 */
uint16_t ATEMext::getAudioMixerMonitorSoloInput() {
    return atemAudioMixerMonitorSoloInput;
}

/**
 * Get Audio Mixer Monitor; Dim
 */
bool ATEMext::getAudioMixerMonitorDim() {
    return atemAudioMixerMonitorDim;
}

/**
 * Set Audio Mixer Monitor; Monitor Audio
 * monitorAudio     Bit 0: On/Off
 */
void ATEMext::setAudioMixerMonitorMonitorAudio(bool monitorAu
dio) {

    _prepareCommandPacket(("CAMm"),12);

        // Set Mask: 1
    _packetBuffer[12+_cBBO+4+4+0] |= 1;

    _packetBuffer[12+_cBBO+4+4+1] = monitorAudio;

    _finishCommandPacket();

}

/**
 * Set Audio Mixer Monitor; Volume
 * volume   0-65381: (DB)
 */
void ATEMext::setAudioMixerMonitorVolume(uint16_t volume) {

    _prepareCommandPacket(("CAMm"),12);

        // Set Mask: 2
    _packetBuffer[12+_cBBO+4+4+0] |= 2;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(volume);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(volume);

    _finishCommandPacket();

}
```

```
/**
 * Set Audio Mixer Monitor; Mute
 * mute     Bit 0: On/Off
 */
void ATEMext::setAudioMixerMonitorMute(bool mute) {

    _prepareCommandPacket(("CAMm"),12);

        // Set Mask: 4
    _packetBuffer[12+_cBBO+4+4+0] |= 4;

    _packetBuffer[12+_cBBO+4+4+4] = mute;

    _finishCommandPacket();

}

/**
 * Set Audio Mixer Monitor; Solo
 * solo     Bit 0: On/Off
 */
void ATEMext::setAudioMixerMonitorSolo(bool solo) {

    _prepareCommandPacket(("CAMm"),12);

        // Set Mask: 8
    _packetBuffer[12+_cBBO+4+4+0] |= 8;

    _packetBuffer[12+_cBBO+4+4+5] = solo;

    _finishCommandPacket();

}

/**
 * Set Audio Mixer Monitor; Solo Input
 * soloInput    (See audio source list)
 */
void ATEMext::setAudioMixerMonitorSoloInput(uint16_t soloInpu
t) {

    _prepareCommandPacket(("CAMm"),12);

        // Set Mask: 16
    _packetBuffer[12+_cBBO+4+4+0] |= 16;

    _packetBuffer[12+_cBBO+4+4+6] = highByte(soloInput);
    _packetBuffer[12+_cBBO+4+4+7] = lowByte(soloInput);
```

```
    _finishCommandPacket();

}

/**
 * Set Audio Mixer Monitor; Dim
 * dim  Bit 0: On/Off
 */
void ATEMext::setAudioMixerMonitorDim(bool dim) {

    _prepareCommandPacket(("CAMm"),12);

        // Set Mask: 32
    _packetBuffer[12+_cBBO+4+4+0] |= 32;

    _packetBuffer[12+_cBBO+4+4+8] = dim;

    _finishCommandPacket();

}

/**
 * Set Audio Levels; Enable
 * enable   Bit 0: On/Off
 */
void ATEMext::setAudioLevelsEnable(bool enable) {

    _prepareCommandPacket(("SALN"),4);

    _packetBuffer[12+_cBBO+4+4+0] = enable;

    _finishCommandPacket();

}

/**
 * Get Audio Mixer Levels; Sources
 */
uint16_t ATEMext::getAudioMixerLevelsSources() {
    return atemAudioMixerLevelsSources;
}

/**
 * Get Audio Mixer Levels; Master Left
 */
int32_t ATEMext::getAudioMixerLevelsMasterLeft() {
    return atemAudioMixerLevelsMasterLeft;
}
```

```
/**
 * Get Audio Mixer Levels; Master Right
 */
int32_t ATEMext::getAudioMixerLevelsMasterRight() {
    return atemAudioMixerLevelsMasterRight;
}

/**
 * Get Audio Mixer Levels; Master Peak Left
 */
int32_t ATEMext::getAudioMixerLevelsMasterPeakLeft() {
    return atemAudioMixerLevelsMasterPeakLeft;
}

/**
 * Get Audio Mixer Levels; Master Peak Right
 */
int32_t ATEMext::getAudioMixerLevelsMasterPeakRight() {
    return atemAudioMixerLevelsMasterPeakRight;
}

/**
 * Get Audio Mixer Levels; Monitor
 */
int32_t ATEMext::getAudioMixerLevelsMonitor() {
    return atemAudioMixerLevelsMonitor;
}

/**
 * Get Audio Mixer Levels; Source Order
 * sources  0-24: Number of
 */
uint16_t ATEMext::getAudioMixerLevelsSourceOrder(uint16_t sou
rces) {
    return atemAudioMixerLevelsSourceOrder[sources];
}

/**
 * Get Audio Mixer Levels; Source Left
 * sources  0-24: Number of
 */
int32_t ATEMext::getAudioMixerLevelsSourceLeft(uint16_t sourc
es) {
    return atemAudioMixerLevelsSourceLeft[sources];
}

/**
 * Get Audio Mixer Levels; Source Right
```

```
 * sources  0-24: Number of
 */
int32_t ATEMext::getAudioMixerLevelsSourceRight(uint16_t sour
ces) {
    return atemAudioMixerLevelsSourceRight[sources];
}

/**
 * Get Audio Mixer Levels; Source Peak Left
 * sources  0-24: Number of
 */
int32_t ATEMext::getAudioMixerLevelsSourcePeakLeft(uint16_t s
ources) {
    return atemAudioMixerLevelsSourcePeakLeft[sources];
}

/**
 * Get Audio Mixer Levels; Source Peak Right
 * sources  0-24: Number of
 */
int32_t ATEMext::getAudioMixerLevelsSourcePeakRight(uint16_t
sources) {
    return atemAudioMixerLevelsSourcePeakRight[sources];
}

/**
 * Set Reset Audio Mixer Peaks; Input Source
 * inputSource  (See audio source list)
 */
void ATEMext::setResetAudioMixerPeaksInputSource(uint16_t inp
utSource) {

    _prepareCommandPacket(("RAMP"),8);

        // Set Mask: 2
    _packetBuffer[12+_cBBO+4+4+0] |= 2;

    _packetBuffer[12+_cBBO+4+4+2] = highByte(inputSource);
    _packetBuffer[12+_cBBO+4+4+3] = lowByte(inputSource);

    _finishCommandPacket();

}

/**
 * Set Reset Audio Mixer Peaks; Master
 * master   Bit 0: Yes/No
 */
void ATEMext::setResetAudioMixerPeaksMaster(bool master) {
```

```
    _prepareCommandPacket(("RAMP"),8);

        // Set Mask: 4
    _packetBuffer[12+_cBBO+4+4+0] |= 4;

    _packetBuffer[12+_cBBO+4+4+4] = master;

    _finishCommandPacket();

}

/**
 * Get Audio Mixer Tally; Sources
 */
uint16_t ATEMext::getAudioMixerTallySources() {
    return atemAudioMixerTallySources;
}

/**
 * Get Audio Mixer Tally; Audio Source
 * sources  0-24: Number of
 */
uint16_t ATEMext::getAudioMixerTallyAudioSource(uint16_t sour
ces) {
    return atemAudioMixerTallyAudioSource[sources];
}

/**
 * Get Audio Mixer Tally; IsMixedIn
 * sources  0-24: Number of
 */
bool ATEMext::getAudioMixerTallyIsMixedIn(uint16_t sources) {
    return atemAudioMixerTallyIsMixedIn[sources];
}

/**
 * Get Tally By Index; Sources
 */
uint16_t ATEMext::getTallyByIndexSources() {
    return atemTallyByIndexSources;
}

/**
 * Get Tally By Index; Tally Flags
 * sources  0-20: Number of
 */
uint8_t ATEMext::getTallyByIndexTallyFlags(uint16_t sources)
{
```

```
    return atemTallyByIndexTallyFlags[sources];
}

/**
 * Get Tally By Source; Sources
 */
uint16_t ATEMext::getTallyBySourceSources() {
    return atemTallyBySourceSources;
}

/**
 * Get Tally By Source; Video Source
 * sources  0-41: Number of
 */
uint16_t ATEMext::getTallyBySourceVideoSource(uint16_t source
s) {
    return atemTallyBySourceVideoSource[sources];
}

/**
 * Get Tally By Source; Tally Flags
 * sources  0-41: Number of
 */
uint8_t ATEMext::getTallyBySourceTallyFlags(uint16_t sources)
 {
    return atemTallyBySourceTallyFlags[sources];
}

/**
 * Get Last State Change Time Code; Hour
 */
uint8_t ATEMext::getLastStateChangeTimeCodeHour() {
    return atemLastStateChangeTimeCodeHour;
}

/**
 * Get Last State Change Time Code; Minute
 */
uint8_t ATEMext::getLastStateChangeTimeCodeMinute() {
    return atemLastStateChangeTimeCodeMinute;
}

/**
 * Get Last State Change Time Code; Second
 */
uint8_t ATEMext::getLastStateChangeTimeCodeSecond() {
    return atemLastStateChangeTimeCodeSecond;
}
```

```
/**
 * Get Last State Change Time Code; Frame
 */
uint8_t ATEMext::getLastStateChangeTimeCodeFrame() {
    return atemLastStateChangeTimeCodeFrame;
}
```

Annex 6

provaV2.cpp

```
#include <iostream>
#include "ATEMstd.cpp"
#include <unistd.h>
#include <time.h>
using namespace std;
void Start();
int main()
{
  int n=0;
  string ip="192.168.10.241";
  char *a= &ip[0];
  ATEMstd c;
  c.begin(a);
  int delay=5000;
  while(1)
  {

    Start();

    c.runLoop();
    if(n==10){
      cout <<"------ "<<"format_I: " <<unsigned(c.getVideoModeFormat())<< "-------";
      c.setVideoModeFormat(9);
    }
    else if(n==11){
      cout <<"------ "<<"format_F: " <<unsigned(c.getVideoModeFormat())<< "-------";
    }
    n+=1;
    usleep(delay);
  }


  return 0;
}
void Start()
{
  cout<<endl <<endl <<" (Start) ";
}
```