```
#ifndef AUNIT_H
#define AUNIT_H

#include <math.h>
#include <QObject>

#include "../SMsg.h"
#include "../Stars.h"

class AUnit0 : public QObject
{
  Q_OBJECT

 protected:
  Stars *s;
  bool Enable;          // if the unit is enable on Stars server or not
  int ALine;            // the line number where the definition appears in .def file

  int LocalStage;

  QString GType;        // Motor, Sensor
  QString Type;         // PM, PZ, ENC, ...
  QString Uid;          // Uniq ID
  QString ID;           // MainTh, StageX, General, ...
  QString Name;         // Displayed name
  QString Dev;
  QString Ch;
  QString DevCh;        // Dev + "." + Ch
  QString Unit;         // metric unit "mm", "mA", ...
  double UPP;

  QString LastFunc;     // last function which enabled isBusy
  QString LastFunc2;    // last function which enabled isBusy2

  bool IsBusy;
  bool IsBusy2;
  int Busy2Count;

  QString Value;
  QStringList Values;
  QString LastValue;
  int ILastSetV;
  double DLastSetV;

  bool HasParent;
  AUnit0 *TheParent;
  QString PUid;

  bool Has2ndDev;
  QString Uid2;
  QString Dev2;
  QString Ch2;
  QString DevCh2;       // Dev + "." + Ch
  AUnit0 *The2ndDev;

public:
  AUnit0( QObject *parent = 0 );

  void Initialize( Stars *s );
  virtual void init( void ) {};

  void setEnable( bool enable );
  virtual void _setEnable( bool /*enable*/ ) {};
  void setALine( int aline ) { ALine = aline; };
  int aLine( void ) { return ALine; };
  bool isEnable( void ) { return Enable; };
```

```
  Stars *getStars( void ) { return s; };
  QString gType( void ) { return GType; };        // Motor, Sensor
  QString type( void ) { return Type; };          // PM, PZ, ENC, ...
  QString uid( void ) { return Uid; };            // Uniq Uid
  QString uid2( void ) { return Uid2; };         /* return Uid2; */  // 2nd Uid
  QString id( void ) { return ID; };              // MainTh, StageX, General, ...
  QString name( void ) { return Name; };          // Displayed name
  QString dev( void ) { return Dev; };
  QString ch( void ) { return Ch; };
  QString devCh( void ) { return DevCh; };        // Driver + "." + Ch
  QString unit( void ) { return Unit; };          // metric unit "mm", "mA", ...
  double upp( void ) { return UPP; };

  void setStars( Stars *S ) { s = S; };
  void setGType( QString gtype ) { GType = gtype; }; // Motor, Sensor
  void setType( QString type ) { Type = type; };     // PM, PZ, ENC, ...
  void setUid( QString uid ) { Uid = uid; };         // Uniq Uid
  void set2ndUid( QString uid ) { Uid2 = uid; };
  void setID( QString id ) { ID = id; };
  void setName( QString name ) { Name = name; };
  void setDev( QString dev )
  { Dev = dev; if ( Ch != "" ) DevCh = makeDevCh( Dev, Ch ); };
  void setCh( QString ch )
  { Ch = ch; if ( Dev != "" ) DevCh = makeDevCh( Dev, Ch ); };
  void setUnit( QString unit ) { Unit = unit; };
  void setUPP( QString upp ) { UPP = upp.toDouble(); };

  QString makeDevCh( const QString &dev, const QString &ch );

  virtual void AskIsBusy( void ) {};
  void setIsBusy( bool busy ) { IsBusy = busy; emit ChangedIsBusy1( Dev ); };
  void setIsBusy2( bool busy ) { IsBusy2 = busy; emit ChangedIsBusy2( Dev ); };
  bool isBusy0( void ) { return IsBusy || IsBusy2; };
  bool isBusy( void ) { return IsBusy; };
  bool isBusy2( void ) { return IsBusy2; };
  void IsBusy2On( QString drv, QString name );
  void IsBusy2Off( QString drv );
  void setBusy2Count( int i ) { Busy2Count = i; };
  void clrBusy2Count( void ) { Busy2Count = 0; };
  int busy2Count( void ) { return Busy2Count; };

  virtual void SetValue( double /* v */ ) {};
  virtual bool GetValue0( void ) { return false; };
  virtual bool GetValue02( void ) { return false; };
  virtual bool GetValue( void );
  QString value( void ) { return Value; };
  QString lastValue( void ) { return LastValue; };
  void setLastValue( QString v ) { LastValue = v; };
  QStringList values( void ) { return Values; };

  void InitLocalStage( void ) { LocalStage = 0; };

  // parent : 完全に定義され Uid を持つデバイス「親デバイス」
  // 例えば nct08 の複数のチャンネルはバラバラにカウントスタートストップできない。
  // ch1 以外のチャンネルのスタートストップも全部 ch1 にまとめて任せるために
  // ch1 を parent とし、他の ch は全部(ch1自信も) ch1 を親に持つ
  void setHasParent( bool hasParent ) { HasParent = hasParent; };
  bool hasParent( void ) { return HasParent; };
  void setPUid( QString puid ) { PUid = puid; };
  QString pUid( void ) { return PUid; };
  //  void setParent( QString pUid ) { PUid = pUid; };
  void setTheParent( AUnit0 *p ) { TheParent = p; };
  AUnit0 *theParent( void ) { return TheParent; };

  // Dev2, Ch2, DevCh2 : ユニットの定義に2つのドライバが必要な時、その2つ目のドライバ
  // 例えばKeithley を電流/電圧アンプとして使用して、その出力を例えばカウンタで図る場
合
  // Keithley の方をコントロールするとレンジ設定ができるカウンタのように扱える
  // この時メインのデバイスはカウンタだが、Keithley を2ndドライバとして指定する
  void setHas2ndDev( bool has2ndDev ) { Has2ndDev = has2ndDev; };
  bool has2ndDev( void ) { return Has2ndDev; };
  AUnit0 *the2ndDev( void ) { return The2ndDev; };
  void setThe2ndDev( AUnit0 *dev2 ) { The2ndDev = dev2; };
  QString dev2( void ) { return Dev2; };
  QString ch2( void ) { return Ch2; };
  QString devCh2( void ) { return DevCh2; };
  void setDev2( QString dev )
  { Dev2 = dev; if ( Ch2 != "" ) DevCh2 = makeDevCh( Dev2, Ch2 ); };
  void setCh2( QString ch )
  { Ch2 = ch; if ( Dev2 != "" ) DevCh2 = makeDevCh( Dev2, Ch2 ); };

  QString lastFunc( void ) { return LastFunc; };
  QString lastFunc2( void ) { return LastFunc2; };

 private slots:
  virtual void ReceiveValues( SMsg msg );
  virtual void SetIsBusyByMsg( SMsg /* msg */ ) {};
  void ClrBusy( SMsg msg );
  virtual void SetCurPos( SMsg /* msg */ );

 signals:
  void ChangedIsBusy1( QString Drv );
  void ChangedIsBusy2( QString Drv );
  void ChangedBusy2Count( QString Drv );
  void Enabled( QString Dev, bool enable );
  void newValue( QString value );
  void newQData( void );
  void LogMsg( QString msg );

#if 0
  //  void CountFinished( void );
  //  void newValues( void );
  void newCountsInROI( QVector<int> );
  void newCountsAll( QVector<int> );
  void newTotalEvents( QVector<int> );
  void newICRs( QVector<double> );
  //  void newDataPoints( int points );
  void Alarm( QString uid, QString msg );
#endif
};

#endif
```

```
#include "AUnit0.h"

AUnit0::AUnit0( QObject *parent ) : QObject( parent )
{
  s = NULL;
  Enable = false;
  ALine = -1;

  GType = "";
  Type = "";
  ID = "";
  Uid = "";
  Name = "";
  Dev = "";
  Ch = "";
  DevCh = "";
  Unit = "";
  UPP = 1;

  HasParent = false;
  TheParent = NULL;
  PUid = "";

  Has2ndDev = false;
  Uid2 = "";
  Dev2 = "";
  Ch2 = "";
  DevCh2 = "";       // Dev + "." + Ch
  The2ndDev = NULL;

  LocalStage = 0;

  IsBusy = IsBusy2 = false;
  Busy2Count = 0;
  IsBusy2Off( "" );

  LastFunc = "";
  LastFunc2 = "";

  Value = "";
  Values.clear();
  LastValue = "";
  ILastSetV = 0;
  DLastSetV = 0;
}

void AUnit0::Initialize( Stars *S )
{
  s = S;

  // 何らかのコマンドに対する応答がエラーだった場合の対処。
  // 単に、isBusy2 をクリアしているだけ。
  // こんなに単純でいいかどうかは難しいところだけれど、
  // enable をちゃんと管理するようにしたので、変な処理に突入することはそちらで避けて
  // 変な処理に突入してしまった場合は、緊急避難的にこの方法で逃げることにする。
  connect( s, SIGNAL( ReceiveError( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );

  // LSR には不要な初期化だが悪くもないので外さない
  connect( s, SIGNAL( AnsIsBusy( SMsg ) ), this, SLOT( SetIsBusyByMsg( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( EvIsBusy( SMsg ) ), this, SLOT( SetIsBusyByMsg( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGetValue( SMsg ) ),this, SLOT( SetCurPos( SMsg ) ),
           Qt::UniqueConnection );
  s->SendCMD2( "Init", "System", "flgon", Dev );
```

```
  s->SendCMD2( "Init", "System", "flgon", DevCh );

  init();   // 各ユニットに固有の処理

  if ( ID == "THETA" ) {
    AskIsBusy();
    GetValue();
  }
  if ( ID == "TotalF" ) {
    connect( s, SIGNAL( AnsGetMCALength( SMsg ) ), this, SLOT( getMCALength( SMsg ) )
,
             Qt::UniqueConnection );
    s->SendCMD2( "SetUpMCA", Dev, "GetMCALength" );
  }
  if ( ID == "ENCTH" ) {
    GetValue();
  }
  emit ChangedIsBusy1( Dev );    // ここの3つのエミットは念の為
  emit ChangedIsBusy2( Dev );
  emit ChangedBusy2Count( Dev );
}

QString AUnit0::makeDevCh( const QString &dev, const QString &ch )
{
  if ( dev == "" ) {
    return "";
  }
  if ( ch != "" )
    return dev + "." + ch;

  return dev;
}

void AUnit0::IsBusy2On( QString drv, QString name )
{
  IsBusy2 = true;
  Busy2Count++;
  LastFunc2 = name;
  emit ChangedIsBusy2( drv );
  emit ChangedBusy2Count( drv );
}

void AUnit0::IsBusy2Off( QString drv )
{
  IsBusy2 = false;
  Busy2Count--;
  if ( Busy2Count < 0 ) Busy2Count = 0;
  LastFunc2 = "";
  emit ChangedIsBusy2( drv );
  emit ChangedBusy2Count( drv );
}

void AUnit0::setEnable( bool enable )
{
  Enable = enable;
  IsBusy = false;
  LastFunc = "";
  _setEnable( enable );    // AUnit0 を継承したクラスでの処理用 // AUnitXMAP が呼んで
る
  emit Enabled( Dev, enable );
  emit ChangedIsBusy1( Dev );
  IsBusy2Off( "" );
}

bool AUnit0::GetValue( void )
{
  IsBusy2On( Dev, "GetValue" );
  s->SendCMD2( Uid, DevCh, "GetValue" );

  return false;
}

void AUnit0::ReceiveValues( SMsg msg )
{
  QString buf;

  if ( ( msg.From() == Dev ) && ( msg.Msgt() == GETVALUES ) ) { // Check !!!!! DevCh/
Drv
    Value = msg.Vals().at(0);
    Values = msg.Vals();

    emit newValue( Value );
    IsBusy2Off( Dev );
  }
}

void AUnit0::ClrBusy( SMsg msg )
{
  if ( ( msg.From() == DevCh ) || ( msg.From() == Dev ) ) {
    IsBusy2Off( Dev );
  }
  if ( Has2ndDev ) {
    if ( ( msg.From() == DevCh2 ) || ( msg.From() == Dev2 ) ) {
      IsBusy2Off( Dev2 );
    }
  }
}

void AUnit0::SetCurPos( SMsg msg )
{
  QString buf;

  if ( ( msg.From() == DevCh )
       && ( ( msg.Msgt() == GETVALUE ) || ( msg.Msgt() == EvCHANGEDVALUE )
            || ( msg.Msgt() == READ ) ) ) {
    Value = msg.Val();
    emit newValue( Value );
    IsBusy2Off( Dev );
  }
}
```

```
#ifndef AMOTOR_H
#define AMOTOR_H

#include "../RelAbs.h"
#include "AUnit0.h"

enum MSPEED { LOW, MIDDLE, HIGH, MSPEEDS };
struct MSPEEDD {
  MSPEED MSid;
  const QString MSName;
};

const MSPEEDD MSpeeds[ MSPEEDS ] = {
  { LOW,    "Low" },
  { MIDDLE, "Middle" },
  { HIGH,   "High" },
};

class AMotor : public AUnit0
{
  Q_OBJECT

 protected:
  double Center;
  bool IsInt;

  bool HasSpeedsLine;
  bool HasSetMaxS;
  int MaxS;      // max speed (pps) 最初に設定されていたオリジナルのスピード
  int MaxMaxS;   // 許される最高のスピード
  int HighS;
  int MiddleS;
  int LowS;

 public:
  AMotor( void );

  void init( void );
  virtual void init0( void ) {};

  virtual void Stop( void ) {};
  bool checkNewVal( void )
  {
    bool rv = ( value() != lastValue() );
    setLastValue( value() );
    return rv;
  };

  double u2p( double u ) { return u / UPP + Center; };
  double p2u( double p ) { return ( p - Center ) * UPP; };
  double any2p( double a, int selU, RELABS ra ) {
    return a / ( ( selU == 0 ) ? 1 : UPP )
      + ( ( ra == REL ) ? Value.toDouble() : ( ( selU == 0 ) ? 0 : Center ) );
  }
  double getCenter( void ) { return Center; };
  void setCenter( QString center ) { Center = center.toDouble(); };
  void setCenter( double center ) { Center = center; };
  double metricValue( void ) { return ( Value.toDouble() - Center ) * UPP; };
  void setIsInt( bool isInt ) { IsInt = isInt; };
  bool isInt( void ) { return IsInt; };

  /* SPeed 設定関連 */
  virtual void SetSpeed( MSPEED /* speed */ ) {};
  virtual void SetHighSpeed( int /* speed */ ) {};
  virtual void SetMiddleSpeed( int /* speed */ ) {};
```

```
  virtual void SetLowSpeed( int /* speed */ ) {};
  bool hasSpeedsLine( void ) { return HasSpeedsLine; };
  void setHasSpeedsLine( bool f ) { HasSpeedsLine = f; };
  int highSpeed( void ) { return MaxS; };     // オリジナルのハイスピード
  void setHighSpeed( int h ) { MaxS = h; };
  int highestSpeed( void ) { return ( HasSpeedsLine ) ? MaxMaxS : MaxS; };// 許容最高
速度
  void setHighestSpeed( int h ) { MaxMaxS = h; };
  int getHighS( void ) { return HighS; };
  int getMiddleS( void ) { return MiddleS; };
  int getLowS( void ) { return LowS; };
  virtual void AskHighSpeed( void ) {};
  virtual void AskMiddleSpeed( void ) {};
  virtual void AskLowSpeed( void ) {};

  /* MStab 可の Unit は下記の関数を実装するべし */
  virtual void CloseShutter( bool /*close*/ ) {};
  virtual void GoMaxAbs( double /*start*/, double /*end*/, int /*steps*/ ) {};
  virtual void GoMaxAbsQ( double /*start*/, double /*end*/, int /*steps*/, double /*t
ime*/ ) {};
  virtual void GoMaxRel( double /*width*/, int /*steps*/ ) {};
  virtual void GoMaxRelQ( double /*width*/, int /*steps*/, double /*time*/ ) {};

  /* PM16C だけかも */
  virtual void AssignDispCh( int /* ch */ ) {};  // ch : 0 - 3 --> 'A' -- 'D'
  /* PM16C で QXAFS の為に */
  virtual void SetTimingOutMode( int /* mode */ ) {};
  // 0 - 5 :: 0: none, 1: cont., 2: 200ns, 3: 10us, 4: 100us, 5: 1ms
  // 34410 triggers rising edge and requires 1us or longer
  // for EB741 2us is long enough
  virtual void SetTimingOutStart( int /*startP*/ ) {};      // start pos. of timing o
ut
  virtual void SetTimingOutEnd( int /*endP*/ ) {};          // end pos. of timing out
  virtual void SetTimingOutInterval( int /*interval*/ ) {}; // timing out interval
  virtual void SetTimingOutReady( int /*ready*/ ) {};       // timing out ready
  virtual int accRate( void ) { return 1; };                // 加減速レート
  virtual void setAccRate( int /*r*/ ) {};
  virtual int accRateNo( void ) { return 0; };              // 加減速レートのテーブル
番号
  virtual void setAccRateNo( int /*n*/ ) {};

  virtual void setMaxV( QString /*maxv*/ ) {};
  virtual void setMinV( QString /*minv*/ ) {};

 public slots:
};

#endif
```

```
#include "AMotor.h"

AMotor::AMotor( void )
{
  Center = 0;

  HasSpeedsLine = false;
  HasSetMaxS = false;
  MaxS = 3000;      // max speed (pps) 最初に設定されていたオリジナルのスピード
  MaxMaxS = 3000;   // 許される最高のスピード
  HighS = 3000;
  MiddleS = 1000;
  LowS = 500;
  IsInt = false;
}

void AMotor::init( void )
{
  // SetValue は Ok: でも Er: でも無視する。(ClrBusyもしない)
  connect( s, SIGNAL( EvChangedValue( SMsg ) ), this, SLOT( SetCurPos( SMsg ) ),
           Qt::UniqueConnection );

  s->SendCMD2( "Init", DevCh, "GetValue" );

  init0();
}
```

```
#ifndef ASENSOR_H
#define ASENSOR_H

#include "AUnit0.h"

class ASensor : public AUnit0
{
  Q_OBJECT

 protected:
  double setTime;       // Actually set time;
  double setDarkTime;   // Actually set time;
  double Dark;                 // back ground value normalized for 1 sec

  bool autoRange;
  bool RangeSelectable; // if range is selectable for the unit
  int RangeU;           // Upper range limit
  int RangeL;           // Lower range limit
  int SelectedRange;

 public:
  ASensor( void );

  void init( void );
  virtual void init0( void ) {};

  virtual bool InitSensor( void ) { return true; };
  virtual double SetTime( double dtime ) { return dtime; };   // in sec
  double GetSetTime( void ) { return setTime; };   // actual set time
  void setDark( double dark ) { Dark = dark; emit newDark( Dark ); };
  double getDark( void ) { return Dark; };

  /* AutoRange 可能なデバイスでは true */
  virtual bool GetRange( void ) { return false; };
  virtual void SetRange( int /* range */ ) {};
  virtual bool isAutoRangeAvailable( void ) { return false; };
  bool isAutoRange( void ) { return autoRange; };
  void setAutoRange( bool ar ) { autoRange = ar; };
  int getRangeU( void ) { return RangeU; };
  int getRangeL( void ) { return RangeL; };
  int getRange( void ) { return SelectedRange; };
  void setRangeSelectable( bool f ) { RangeSelectable = f; };
  bool isRangeSelectable( void ) { return RangeSelectable; };
  void setRange( int r ) { SelectedRange = r; };
  void setRangeU( int upper ) { RangeU = upper; };
  void setRangeL( int lower ) { RangeL = lower; };

  /* QXAFS 対応 */
  virtual bool QStart( void ) { return false; };
  virtual bool QRead( void ) { return false; };
  virtual bool QEnd( void ) { return false; };

  /* 連続スキャン対応 */
  virtual bool Close( void ) { return false; };

 signals:
  void newDark( double dark );
  void AskedNowRange( int r );

};

#endif
```

```
#include "ASensor.h"

ASensor::ASensor( void )
{
  setTime = 1;       // Actually set time;
  setDarkTime = 1;   // Actually set time;
  Dark = 0;          // back ground value normalized for 1 sec

  autoRange = false;
  RangeSelectable = false;
  RangeU = 0;           // Upper range limit
  RangeL = 0;           // Lower range limit
  SelectedRange = 0;
}

void ASensor::init( void )
{
  init0();
}
```

```
#ifndef AUNITPM_H
#define AUNITPM_H

#include "AMotor.h"

class AUnitPM : public AMotor
{
  Q_OBJECT

  int AccRate;   // 加減速レート ( AccRage ms/1000pps )
  int AccRateNo; // 対応する加減速レートの PM16C のテーブル番号

 public:
  AUnitPM( void );
  void init0( void );

  void SetValue( double v );
  void SetSpeed( MSPEED speed );
  void SetHighSpeed( int speed );
  void SetMiddleSpeed( int speed );
  void SetLowSpeed( int speed );
  void Stop( void );
  void AskHighSpeed( void );
  void AskMiddleSpeed( void );
  void AskLowSpeed( void );
  void AssignDispCh( int ch );
  void SetTimingOutMode( int mode );
  void SetTimingOutStart( int startP );
  void SetTimingOutEnd( int endP );
  void SetTimingOutInterval( int interval );
  void SetTimingOutReady( int ready );
  void AskIsBusy( void );

  int accRate( void ) { return AccRate; };    // 加減速レート
  void setAccRate( int r ) { AccRate = r; };
  int accRateNo( void ) { return AccRateNo; }; // 加減速レートのテーブル番号
  void setAccRateNo( int n ) { AccRateNo = n; };

 private slots:
  void SetIsBusyByMsg( SMsg msg );
  void RcvHighSpeed( SMsg msg );
  void RcvMiddleSpeed( SMsg msg );
  void RcvLowSpeed( SMsg msg );
  //  void RcvAnsGetValueOfDriver( SMsg msg );

 signals:
  void gotHighS( int s );
  void gotMiddleS( int s );
  void gotLowS( int s );

};

#endif
```

```
#include "AUnitPM.h"

AUnitPM::AUnitPM( void )
{
  AccRate = 100;    // 加減速レート ( AccRage ms/1000pps )
  AccRateNo = 24;   // 対応する加減速レートの PM16C のテーブル番号
}

void AUnitPM::init0( void )
{
  s->SendCMD2( "Init", DevCh, "IsBusy" );

  connect( s, SIGNAL(AnsSetHighSpeed( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsSetMiddleSpeed( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsSetLowSpeed( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsSetTimingOutMode( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsSetTimingOutStart( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsSetTimingOutEnd( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsSetTimingOutInterval( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsSetTimingOutReady( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsSelect( SMsg )), this, SLOT(ClrBusy( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsGetHighSpeed( SMsg )), this, SLOT(RcvHighSpeed( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsGetMiddleSpeed( SMsg )), this, SLOT(RcvMiddleSpeed( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsGetLowSpeed( SMsg )), this, SLOT(RcvLowSpeed( SMsg )),
           Qt::UniqueConnection );
}

void AUnitPM::SetValue( double v )
{
  IsBusy = true;
  emit ChangedIsBusy1( DevCh );
  s->SendCMD2( Uid, DevCh, "SetValue", QString::number( ILastSetV = (int)v ) );
}

void AUnitPM::SetSpeed( MSPEED speed )
{
  QString cmd = "SpeedLow";

  //  IsBusy2 = true;    // setspeed に対する応答は無視するので isBusy2 もセットしな
い
  switch( speed ) {
  case LOW: cmd = "SpeedLow"; break;
  case MIDDLE: cmd = "SpeedMiddle"; break;
  case HIGH: cmd = "SpeedHigh"; break;
  default: cmd = "SpeedLow"; break;
  }
  s->SendCMD2( Uid, DevCh, cmd );
}

void AUnitPM::SetHighSpeed( int speed )
{
  IsBusy2On( Dev, "SetHighSpeed" );
  QString cmd = QString( "SetHighSpeed %1" ).arg( speed );
  s->SendCMD2( Uid, DevCh, cmd );
```

```
}

void AUnitPM::SetMiddleSpeed( int speed )
{
  IsBusy2On( Dev, "SetMiddleSpeed" );
  QString cmd = QString( "SetMiddleSpeed %1" ).arg( speed );
  s->SendCMD2( Uid, DevCh, cmd );
}

void AUnitPM::SetLowSpeed( int speed )
{
  IsBusy2On( Dev, "SetLowSpeed" );
  QString cmd = QString( "SetLowSpeed %1" ).arg( speed );
  s->SendCMD2( Uid, DevCh, cmd );
}

void AUnitPM::Stop( void )
{
  s->SendCMD2( Uid, DevCh, "Stop" );
}

void AUnitPM::AskHighSpeed( void )
{
  s->SendCMD2( Uid, DevCh, "GetHighSpeed" );
}

void AUnitPM::AskMiddleSpeed( void )
{
  s->SendCMD2( Uid, DevCh, "GetMiddleSpeed" );
}

void AUnitPM::AskLowSpeed( void )
{
  s->SendCMD2( Uid, DevCh, "GetLowSpeed" );
}

void AUnitPM::AssignDispCh( int ch )
{
  IsBusy2On( Dev, "AssignDispCh" );
  int num = Ch.toInt();
  if ( Ch.left( 2 ).toUpper() == "CH" )
    num = Ch.mid( 2 ).toInt();
  QString cmd = QString( "Select %1 %2" ).arg( QChar( 'A' + ch ) ).arg( num );
  s->SendCMD2( Uid, Dev, cmd );
}

void AUnitPM::SetTimingOutMode( int mode )
{
  IsBusy2On( Dev, "SetTimingOutMode" );
  QString cmd = QString( "SetTimingOutMode %1" ).arg( mode );
  s->SendCMD2( Uid, DevCh, cmd );
}

void AUnitPM::SetTimingOutStart( int startP )
{
  IsBusy2On( Dev, "SetTimingOutStart" );
  QString cmd = QString( "SetTimingOutStart %1" ).arg( startP );
  s->SendCMD2( Uid, DevCh, cmd );
}

void AUnitPM::SetTimingOutEnd( int endP )
{
  IsBusy2On( Dev, "SetTimingOutEnd" );
  QString cmd = QString( "SetTimingOutEnd %1" ).arg( endP );
  s->SendCMD2( Uid, DevCh, cmd );
}
```

```
void AUnitPM::SetTimingOutInterval( int interval )
{
  IsBusy2On( Dev, "SetTimingOutInterval" );
  QString cmd = QString( "SetTimingOutInterval %1" ).arg( interval );
  s->SendCMD2( Uid, DevCh, cmd );
}

void AUnitPM::SetTimingOutReady( int ready )
{
  IsBusy2On( Dev, "SetTimingOutReady" );
  QString cmd = QString( "SetTimingOutReady %1" ).arg( ready );
  s->SendCMD2( Uid, DevCh, cmd );
}

void AUnitPM::AskIsBusy( void )
{
  s->SendCMD2( Uid, DevCh, "IsBusy" );
}

void AUnitPM::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == DevCh )
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}

void AUnitPM::RcvHighSpeed( SMsg msg )
{
  if ( ( ( msg.From() == DevCh )||( msg.From() == Dev ) )  // Check !!!!! DevCh/Drv
       && ( ( msg.Msgt() == GETHIGHSPEED ) ) ) {
    HighS = msg.Val().toInt();
    if ( ! HasSetMaxS ) {
      MaxS = HighS;
      HasSetMaxS = true;
    }
    IsBusy2Off( Dev );
    emit gotHighS( HighS );
  }
}

void AUnitPM::RcvMiddleSpeed( SMsg msg )
{
  if ( ( ( msg.From() == DevCh )||( msg.From() == Dev ) )  // Check !!!!! DevCh/Drv
       && ( ( msg.Msgt() == GETMIDDLESPEED ) ) ) {
    MiddleS = msg.Val().toInt();
    IsBusy2Off( Dev );
    emit gotMiddleS( MiddleS );
  }
}

void AUnitPM::RcvLowSpeed( SMsg msg )
{
  if ( ( ( msg.From() == DevCh )||( msg.From() == Dev ) )  // Check !!!!! DevCh/Drv
       && ( ( msg.Msgt() == GETLOWSPEED ) ) ) {
    LowS = msg.Val().toInt();
    IsBusy2Off( Dev );
    emit gotLowS( LowS );
  }
}
```

```
#ifndef AUNITSC_H
#define AUNITSC_H

#include "AMotor.h"

class AUnitSC : public AMotor
{
  Q_OBJECT

 public:
  AUnitSC( void );
  void init0( void );

  void SetValue( double v );
  bool GetValue( void );
  void Stop( void );
  void AskIsBusy( void );

 private slots:
  void SetCurPos( SMsg msg );
  void SetIsBusyByMsg( SMsg msg );
};

#endif
```

```
#include "AUnitSC.h"

AUnitSC::AUnitSC( void )
{
}

void AUnitSC::init0( void )
{
  s->SendCMD2( "Init", DevCh, "IsBusy" );
}

void AUnitSC::SetValue( double v )
{
  s->SendCMD2( Uid, DevCh, "SetValue", QString( "%1 1 0 0 0 0" )
               .arg( ILastSetV = (int)v ) );
}

bool AUnitSC::GetValue( void )
{
  IsBusy2On( Dev, "GetValue" );
  s->SendCMD2( Uid, DevCh, "GetValue 0" );
  return false;
}

void AUnitSC::Stop( void )
{
  s->SendCMD2( Uid, DevCh, "Stop" );
}

void AUnitSC::AskIsBusy( void )
{
  s->SendCMD2( Uid, DevCh, "IsBusy" );
}

void AUnitSC::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == DevCh )
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}

void AUnitSC::SetCurPos( SMsg msg )
{
  QString buf;

  if ( ( msg.From() == DevCh )
       && ( ( msg.Msgt() == GETVALUE ) || ( msg.Msgt() == EvCHANGEDVALUE )
            || ( msg.Msgt() == READ ) ) ) {
    if ( msg.Msgt() == EvCHANGEDVALUE ) {
      Value = msg.Val();
    } else {
      Value = msg.Vals().at(1);
    }
    emit newValue( Value );
    IsBusy2Off( Dev );
  }
}
```

```
#ifndef AUNITPZ_H
#define AUNITPZ_H

#include "AMotor.h"

class AUnitPZ : public AMotor
{
  Q_OBJECT

  double MaxV, MinV;

 public:
  AUnitPZ( void );
  void init0( void );

  void setMaxV( QString maxv ) { MaxV = maxv.toDouble(); };
  void setMinV( QString minv ) { MinV = minv.toDouble(); };
};

#endif
```

```
#include "AUnitPZ.h"

AUnitPZ::AUnitPZ( void )
{
  MaxV = 0;          // only for PZ
  MinV = 0;          // only for PZ
}

void AUnitPZ::init0( void )
{
  connect( s, SIGNAL( EvChangedValue( SMsg ) ), this, SLOT( SetCurPos( SMsg ) ),
           Qt::UniqueConnection );

  connect( s, SIGNAL( AnsGoMaxAbs( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGoMaxRel( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsShutterOff( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  s->SendCMD2( "Init", DevCh, "Init" );

}
```

```
#ifndef AUNITCNT_H
#define AUNITCNT_H

#include "ASensor.h"

class AUnitCNT : public ASensor
{
  Q_OBJECT

 public:
  AUnitCNT( void ) {};
  void init0( void );
  virtual void init00( void );
  bool InitSensor( void );
  virtual bool _InitSensor( void );

  void AskIsBusy( void );
  double SetTime( double dtime );
  virtual void _AskIsBusy( void );
  bool GetValue0( void );
  bool GetValue02( void );

  bool Close( void );

 private slots:
  void SetIsBusyByMsg( SMsg msg );
};

class AUnitCNT2 : public AUnitCNT
{
  Q_OBJECT

 public:
  AUnitCNT2( void ) {};

  void init00( void );
  bool _InitSensor( void );

  void _AskIsBusy( void ) {};

  bool isAutoRangeAvailable( void ) { return true; }
  bool GetRange( void );
  void SetRange( int range );

 private slots:
  void ReactGetRange( SMsg msg );

};

#endif
```

```cpp
#include "AUnitCNT.h"

void AUnitCNT::init0( void )
{
  connect( s, SIGNAL( AnsSetStopMode( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetTimerPreset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsCounterReset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsCountStart( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsStop( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  s->SendCMD2( "Init", Dev, "SetStopMode", "T" );

  init00();
}

void AUnitCNT::init00( void )
{
  s->SendCMD2( "Init", Dev, "IsBusy" );
}

void AUnitCNT2::init00( void )
{
  s->SendCMD2( "Init", Dev, "IsBusy" );

  connect( s, SIGNAL( AnsReset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetAutoRange( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetZeroCheck( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetRange( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGetRange( SMsg ) ), this, SLOT( ReactGetRange( SMsg ) ),
           Qt::UniqueConnection );
}

bool AUnitCNT::InitSensor( void )
{
  return _InitSensor();
}

bool AUnitCNT::_InitSensor( void )
{
  return false;
}

bool AUnitCNT2::_InitSensor( void )
{
  if ( The2ndDev == NULL ) {
    qDebug() << "InitSensor: the 2nd Dev is not defined for" << Uid << Name;
    return false;
  }

  bool rv;
  // CNT2, OTC2 のとき カウンタの向こうにつながるのは
  // keithley なのでそれ用の処理をしておく
  QString Type2 = The2ndDev->type();
  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev2, "InitSensor-c0" );
    s->SendCMD2( "Scan", DevCh2, "Reset", "" );
```

```cpp
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev2, "InitSensor-c1" );
    if ( autoRange ) {
      if ( Type2 == "PAM" )
        s->SendCMD2( "Scan", DevCh2, "SetAutoRangeEnable", "1" );
      if ( Type2 == "PAM2" )
        s->SendCMD2( "Scan", Dev2, "SetAutoRangeEnable " + Ch2, "1" );
      LocalStage = 3;
    } else {
      if ( Type2 == "PAM" )
        s->SendCMD2( "Scan", DevCh2, "SetAutoRangeEnable", "0" );
      if ( Type2 == "PAM2" )
        s->SendCMD2( "Scan", Dev2, "SetAutoRangeEnable " + Ch2, "0" );
      LocalStage = 2;
    }
    rv = true;
    break;
  case 2:
    IsBusy2On( Dev2, "InitSensor-c2" );
    if ( Type2 == "PAM" ) {
      s->SendCMD2( "Scan", DevCh2, "SetRange", QString( "2E%1" ).arg( SelectedRange )
 );
      LocalStage++;
      rv = true;
    }
    if ( Type2 == "PAM2" ) {
      s->SendCMD2( "Scan", Dev2, "SetRange " + Ch2,
                   QString( "2E%1" ).arg( SelectedRange ) );
      LocalStage+=2;     // PAM2 の時は、LocalStage == 3 をとばす
      rv = false;
    }
    break;
  case 3:
    IsBusy2On( Dev2, "InitSensor-c3" );
    s->SendCMD2( "Scan", DevCh2, "SetZeroCheckEnable", "0" );
    rv = false;
    LocalStage++;
    break;
  default:
    rv = false;
  }

  return rv;
}

void AUnitCNT::AskIsBusy( void )
{
  _AskIsBusy();   // CNT だけ反応して CNT2 は反応しない
}

void AUnitCNT::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == Dev )   // Check !!!!! DevCh/Drv
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}
```

```
void AUnitCNT::_AskIsBusy( void )
{
  s->SendCMD2( Uid, DevCh, "IsBusy" );
}

bool AUnitCNT::GetValue0( void )  // 値読み出しコマンドの前に何か必要なタイプの場合
{
  bool rv = false;

  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "GetValue0c0" );
    s->SendCMD2( Uid, Dev, "CounterReset" );
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev, "GetValue0c1" );
    IsBusy = true;
    LastFunc = "GetValue0c1";
    emit ChangedIsBusy1( Dev );
    s->SendCMD2( Uid, Dev, "CountStart" );
    LocalStage++;
    rv = false;
    break;
  }

  return rv;
}

// 値読み出しコマンドの前に何か必要なタイプの場合
// 別バージョン、presetTime 等の終了条件無しにしてある
// 連続スキャン (差分で値を見る)モード用
bool AUnitCNT::GetValue02( void )
{
  bool rv = false;

  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "GetValue0c0" );
    s->SendCMD2( Uid, Dev, "SetStopMode", "N" );
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev, "GetValue0c1" );
    s->SendCMD2( Uid, Dev, "CounterReset" );
    LocalStage++;
    rv = true;
    break;
  case 2:
    IsBusy2On( Dev, "GetValue0c2" );
    IsBusy = true;
    LastFunc = "GetValue0c1";
    emit ChangedIsBusy1( Dev );
    s->SendCMD2( Uid, Dev, "CountStart" );
    LocalStage++;
    rv = false;
    break;
  }

  return rv;
}

// 連続スキャンの後にノーマルモードに戻す
bool AUnitCNT::Close( void )
```

```
{
  bool rv = false;

  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "Close0" );
    s->SendCMD2( Uid, Dev, "Stop" );
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev, "Close1" );
    s->SendCMD2( Uid, Dev, "SetStopMode", "T" );
    LocalStage++;
    rv = false;
    break;
  }

  return rv;
}

bool AUnitCNT2::GetRange( void ) // CNT2, OTC2
{
  QString Type2 = The2ndDev->type();
  IsBusy2On( Dev2, "GetRange" );
  if ( Type2 == "PAM" )
    s->SendCMD2( Uid, DevCh2, QString( "GetRange" ) );
  if ( Type2 == "PAM2" )
    s->SendCMD2( Uid, Dev2, QString( "GetRange " ) + Ch2 );

  return false;
}

void AUnitCNT2::ReactGetRange( SMsg msg )  // CNT2, OTC2
{
  double range = RangeL;
  if ( ( msg.From() == DevCh2 ) || ( msg.From() == Dev2 ) ) {
    QString Type2 = The2ndDev->type();
    if ( Type2 == "PAM" ) {
      range = log10( msg.Vals().at(0).toDouble() / 2.1 );
    }
    if ( Type2 == "PAM2" ) {
      if ( msg.Vals().at(0).toInt() == Ch2.toInt() ) {
        range = log10( msg.Vals().at(1).toDouble() / 2.1 );
      } else {
        return;
      }
    }

    IsBusy2Off( Dev2 );
    if ( range > RangeU ) range = RangeU;
    if ( range < RangeL ) range = RangeL;
    emit AskedNowRange( (int)range );
  }
}

double AUnitCNT::SetTime( double dtime ) // in sec // この関数は、複数ステップ化でき
ない
{
  long int ltime;

  IsBusy2On( Dev, "SetTime" );
  ltime = dtime * 1e6;
  s->SendCMD2( Uid, Dev, "SetTimerPreset", QString::number( ltime ) );
  setTime = dtime;
  return setTime;
}

void AUnitCNT2::SetRange( int range )
{
  if ( The2ndDev == NULL ) {
    qDebug() << "SetRange: the 2nd Dev is not defined for" << Uid << Name;
    return;
  }

  IsBusy2On( Dev2, "SetRange" );
  // CNT2, OTC2 のとき カウンタの向こうにつながるのは
  // keithley ( PAM/PAM2 )なのでそれ用の処理をしておく
  QString Type2 = The2ndDev->type();
  if ( Type2 == "PAM" ) {
    s->SendCMD2( "Scan", DevCh2, "SetRange", QString( "2E%1" ).arg( range ) );
  }
  if ( Type2 == "PAM2" ) {
    s->SendCMD2( "Scan", Dev2, "SetRange " + Ch2,
                 QString( "2E%1" ).arg( range ) );
  }
  SelectedRange = range;
}
```

```
#ifndef AUNITDV_H
#define AUNITDV_H

#include "ASensor.h"

class AUnitDV : public ASensor
{
  Q_OBJECT

 protected:
  int points;           // Measured Data Points : 34410
  bool HasMaxIntTime;
  double MaxIntTime;    // Maximum integration time

 public:
  AUnitDV( void );
  void init0( void );
  virtual void init00( void );
  bool InitSensor( void );
  virtual bool _InitSensor( void );

  double SetTime( double dtime );

  bool hasMaxIntTime( void ) { return HasMaxIntTime; };
  double maxIntTime( void ) { return MaxIntTime; };
  void setMaxIntTime( double t ) { MaxIntTime = t; HasMaxIntTime = true; };
  int Points( void ) { return points; };

  bool QStart( void );
  bool QRead( void );
  bool QEnd( void );

  void SetTriggerDelay( double time );
  void SetSamplingSource( QString source );
  void SetTriggerSource( QString source );
  void SetTriggerCounts( int counts );
  void SetTriggerSlope( QString type );
  void GetDataPoints( void );
  void ReadDataPoints( int datas );
  void Abort( void );

 private slots:
  void SetIsBusyByMsg( SMsg msg );
  void RcvQGetData( SMsg msg );
};

class AUnitDV2 : public AUnitDV
{
  Q_OBJECT

 public:
  AUnitDV2( void ) {};
  void init00( void );
  bool _InitSensor( void );

  void AskIsBusy( void );
};

#endif
```

```
#include "AUnitDV.h"

AUnitDV::AUnitDV( void )
{
  points = 0;
  HasMaxIntTime = false;
  MaxIntTime = 1000000;   // 十分大きい
}

void AUnitDV::init0( void )
{
  connect( s, SIGNAL( AnsReset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );

  init00();
}

void AUnitDV::init00( void )
{
  connect( s, SIGNAL( AnsQInitialize( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsQGetData( SMsg ) ), this, SLOT( RcvQGetData( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsQFinalize( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
}

void AUnitDV2::init00( void )
{
  connect( s, SIGNAL( AnsSetAperture( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetAutoZero( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
}

bool AUnitDV::InitSensor( void )
{
  return _InitSensor();
}

bool AUnitDV::_InitSensor( void )
{
  IsBusy2On( Dev, "InitSensor-c0" );
  s->SendCMD2( "Scan", DevCh, "Reset", "" );
  return false;
}

bool AUnitDV2::_InitSensor( void )
{
  bool rv = false;

  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "InitSensor-c0" );
    s->SendCMD2( "Scan", DevCh, "Reset", "" );
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev, "InitSensor-c1" );
    s->SendCMD2( "Scan", DevCh, "SetAutoZero", "OFF" );
    LocalStage++;
    rv = false;
    break;
  default:
```

```
    rv = false;
  }

  return rv;
}

void AUnitDV2::AskIsBusy( void )
{
  s->SendCMD2( Uid, DevCh, "IsBusy" );
}

void AUnitDV::SetIsBusyByMsg( SMsg msg ) // DV, DV2
{
  if ( ( msg.From() == DevCh )
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}


/* DV/DV2 は DV の方が QXAFS 用で、DV2 が NORMAL 用 */

bool AUnitDV::QStart( void )
{
  IsBusy2On( Dev, "Start" );
  s->SendCMD2( Uid, DevCh, "qInitialize", QString::number( setTime ) );
  return false;
}

bool AUnitDV::QRead( void )
{
  IsBusy2On( Dev, "Read" );
  s->SendCMD2( Uid, DevCh, "qGetData" );

  return false;
}

bool AUnitDV::QEnd( void )
{
  IsBusy2On( Dev, "End" );
  s->SendCMD2( Uid, DevCh, "qFinalize" );

  return false;
}

double AUnitDV::SetTime( double dtime ) // in sec // この関数は、複数ステップ化できな
い
{
  if ( dtime < 0.0001 ) dtime = 0.0001;
  if ( dtime > 1.0 ) dtime = 1.0;
  if (( HasMaxIntTime )&&( dtime > MaxIntTime )) { dtime = MaxIntTime; };
  if ( Type == "DV2" ) {   // DV の場合、ここでは内部変数 setTime に値を設定するだけ
。
    IsBusy2On( Dev, "SetAperture" );
    s->SendCMD2( Uid, DevCh, "SetAperture", QString( "%1" ).arg( dtime ) );
  }
  setTime = dtime;

  return setTime;
}
```

```
void AUnitDV::SetTriggerDelay( double time )  // 使っていない
{
  IsBusy2On( Dev, "SetTriggerDelay" );
  s->SendCMD2( Uid, DevCh, "SetTriggerDelay", QString::number( time ) );
}

void AUnitDV::SetSamplingSource( QString source ) //source := TIM, IMM // 使ってない
{
  IsBusy2On( Dev, "SetSamplingSource" );
  s->SendCMD2( Uid, DevCh, "SetSamplingSource", source );
}

void AUnitDV::SetTriggerSource( QString source ) //source := IMM, EXT, BUS //使ってな
い
{
  IsBusy2On( Dev, "SetTriggerSource" );
  s->SendCMD2( Uid, DevCh, "SetTriggerSource", source );
}

void AUnitDV::SetTriggerCounts( int counts )
{
  IsBusy2On( Dev, "SetTriggerCounts" );

  QString arg;
  if ( counts > 0 )
    arg = QString::number( counts );
  else
    arg = "INF";

  s->SendCMD2( Uid, DevCh, "SetTriggerCounts", arg );
}

void AUnitDV::SetTriggerSlope( QString type ) // POS : rising-edge
{
  IsBusy2On( Dev, "SetTriggerSlope" );
  s->SendCMD2( Uid, DevCh, "SetTriggerSlope", type );
}

void AUnitDV::GetDataPoints( void )
{
  IsBusy2On( Dev, "GetDataPoints" );
  s->SendCMD2( Uid, DevCh, "GetDataPoints" );
}

void AUnitDV::ReadDataPoints( int datas )
{
  IsBusy2On( Dev, "ReadDataPoints" );
  s->SendCMD2( Uid, DevCh, "GetDataPoints", QString::number( datas ) );
}

void AUnitDV::Abort( void )
{
  IsBusy2On( Dev, "Abort" );
  s->SendCMD2( Uid, DevCh, "Abort" );
}

void AUnitDV::RcvQGetData( SMsg msg )
{
  if ( ( ( msg.From() == DevCh )||( msg.From() == Dev ) )
       && ( ( msg.Msgt() == GETDATAPOINTS )
            || ( msg.Msgt() == QGETDATA )
            || ( msg.Msgt() == GETDATA ) ) ) {

    Values = msg.Vals();
    emit newQData();
    IsBusy2Off( Dev );
```

```
  }
}
```

```
#ifndef AUNITXMAP_H
#define AUNITXMAP_H

#include "ASensor.h"
#include "XMAPHead.h"

enum STATELM { STAT_REALTIME, STAT_TRG_LIVETIME, STAT_ENGY_LIVETIME, STAT_TRIGGERS,
               STAT_EVENTS, STAT_ICR, STAT_OCR };

#define AXMAPBUF    ( XMAPHEAD + 2048 * 4 ) // MCAHEAD + 2048 pixels * 4byte
#define XMAPBUFSIZE ( AXMAPBUF * 19 )       // AMCABUF * 19 ch

const int MaxSSDs = 19;       // Max SSD elements

class AUnitXMAP : public ASensor
{
  Q_OBJECT

  bool ConnectedToSSDServer;
  bool hasConnected;

  QString DataLinkHostName;
  qint16 DataLinkHostPort;
  QTcpSocket *dLink;
  QDataStream *dLinkStream;
  int dLinkCount;

  char *MCAs0, *MCAs;
  bool MCAsReady;    // MCAs に有効なデータがある true, 無い false

  QString SSDPresetType;
  QString *ROIStart, *ROIEnd;
  QVector<quint64> CountsInROI;
  QVector<quint64> CountsAll;
  QVector<quint64> TotalEvents;
  QVector<double> ICRs;
  QVector<double> DarkCountsInROI;    // per second
  QVector<double> DarkCountsAll;      // per second
  QVector<double> DarkTotalEvents;    // per second
  QVector<double> DarkICRs;           // per second
  quint64 MCALength;
  QStringList MCAStats;
  QVector<double> MCARealTime;
  QVector<double> MCALiveTime;
  QVector<bool> SSDUsingCh;

 public:
  AUnitXMAP( void );

  bool InitSensor( void );
  void init0( void );
  void _setEnable( bool enable );
  void ConnectToXMAPDataLinkServer( QString host, qint16 port );

  bool GetValue0( void );
  bool GetValue02( void );
  void RunStart( void );
  void RunStop( void );
  void RunResume( void );

  void setSSDPresetType( QString type ) { SSDPresetType = type; };
  QString getSSDPresetType( void ) { return SSDPresetType; };

  double SetTime( double dtime );
  bool GetMCAs( void );
```

```
  bool GetStat( void );
  //  bool SetRealTime( double val );  // xmap2
  //  bool SetLiveTime( double val );  // xmap2
  bool SetRealTime( int ch, double val );
  bool SetLiveTime( int ch, double val );
  bool GetRealTime( int ch );
  bool GetLiveTime( int ch );

  void setDark( void );

  QVector<quint64> getCountsInROI( void ) { return CountsInROI; };
  QVector<quint64> getCountsAll( void ) { return CountsAll; };
  QVector<quint64> getTotalEvents( void ) { return TotalEvents; };
  QVector<double> getICRs( void ) { return ICRs; };
  QVector<double> getDarkCountsInROI( void ) { return DarkCountsInROI; };
  QVector<double> getDarkCountsAll( void ) { return DarkCountsAll; };
  QVector<double> getDarkTotalEvents( void ) { return DarkTotalEvents; };
  QVector<double> getDarkICRs( void ) { return DarkICRs; };
  quint32 *getAMCA( int ch );
  quint32 getAMCAdata( int ch, int pixel );
  XMAPHead getAMCAHead( int ch );

  void setGain( int ch, double gain );
  void setROIs( QString *roistart, QString *roiend )
  { ROIStart = roistart; ROIEnd = roiend; };

  void SetLowLimit( int ch, int llpix );
  void setSSDUsingCh( int i, bool f )
  { if ( i < MaxSSDs ) SSDUsingCh[i] = f; };
  bool getSSDUsingCh( int i )
  { if ( i < MaxSSDs ) return SSDUsingCh[i]; else return false; };

  double realTime( int ch );
  double liveTime( int ch );

  double stat( int ch, STATELM i );
  bool GetValue( void );
  bool Close( void );

 public slots:
  void SetIsBusyByMsg( SMsg msg );
  void getMCALength( SMsg msg );
  void ReactGetStat( SMsg msg );
  void ReactGetRealTime( SMsg msg );
  void ReactGetLiveTime( SMsg msg );
  void ReactGetDataLinkCh( SMsg msg );
  void ReceiveValues( SMsg msg );
  void receiveMCAs( void );

 signals:
  void ReceivedNewMCARealTime( int i );
  void ReceivedNewMCALiveTime( int i );
  void DataLinkServerIsReady( QString host, qint16 port );
  void NewMCAsAvailable( char *MCAs );

};

#endif
```

```
#include "AUnitXMAP.h"

AUnitXMAP::AUnitXMAP( void )
{
  MCALength = 2048;    // !!

  ConnectedToSSDServer = false;
  hasConnected = false;
  DataLinkHostName = "";
  DataLinkHostPort = 0;
  dLink = NULL;
  dLinkStream = NULL;
  dLinkCount = 0;

  MCAs0 = NULL;
  MCAs = NULL;
  MCAsReady = false;

  SSDPresetType = "REAL";   // for MCA/SSD
  ROIStart = ROIEnd = NULL;

  MCARealTime.clear();
  MCALiveTime.clear();
  SSDUsingCh.clear();
  CountsInROI.clear();
  CountsAll.clear();
  TotalEvents.clear();
  ICRs.clear();
  DarkCountsInROI.clear();
  DarkCountsAll.clear();
  DarkTotalEvents.clear();
  DarkICRs.clear();
  for ( int i = 0; i < MaxSSDs; i++ ) {
    MCARealTime << 0;
    MCALiveTime << 0;
    SSDUsingCh << true;
    CountsInROI << 0;
    CountsAll << 0;
    TotalEvents << 0;
    ICRs << 0;
    DarkCountsInROI << 0;
    DarkCountsAll << 0;
    DarkTotalEvents << 0;
    DarkICRs << 0;
  }

  MCAStats.clear();
}

bool AUnitXMAP::InitSensor( void )
{
  bool rv = false;

  QString ROIs = "";
  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "InitSensor-c0" );
    s->SendCMD2( "Init", Dev, "RunStop" );
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev, "InitSensor-c1" );
    s->SendCMD2( "Init", Dev, "SetPresetType", SSDPresetType );
    LocalStage++;
```

```
    rv = true;
    break;
  case 2:
    IsBusy2On( Dev, "InitSensor-c2" );
    ROIs = ROIStart[0] + " " + ROIEnd[0];
    for ( int i = 1; i < MaxSSDs; i++ ) {
      ROIs += " " + ROIStart[i] + " " + ROIEnd[i];
    }
    s->SendCMD2( "Init", Dev, "SetROIs", ROIs );
    LocalStage++;
    rv = false;
    break;
  }

  return rv;
}

void AUnitXMAP::init0( void )
{
  connect( s, SIGNAL( AnsSetPresetType( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetPresetValue( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsRunStart( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsRunStop( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGetValues( SMsg ) ), this, SLOT( ReceiveValues( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetROIs( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGetStatistics( SMsg ) ), this, SLOT( ReactGetStat( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGetRealTime( SMsg )), this, SLOT( ReactGetRealTime( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGetLiveTime( SMsg )), this, SLOT( ReactGetLiveTime( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGetDataLinkCh( SMsg ) ),
           this, SLOT( ReactGetDataLinkCh( SMsg ) ),
           Qt::UniqueConnection );

  s->SendCMD2( "Init", DevCh, "IsBusy" );
  s->SendCMD2( "Init", Dev, "RunStop" );
  s->SendCMD2( "Init", Dev, "GetDataLinkCh" );
}

void AUnitXMAP::ConnectToXMAPDataLinkServer( QString host, qint16 port )
{
  if ( !hasConnected ) {
    hasConnected = true;
    qDebug() << "data link server" << host << port;
    if ( dLink != NULL ) delete dLink;
    dLink = new QTcpSocket;
    if ( dLinkStream != NULL ) delete dLinkStream;
    dLinkStream = new QDataStream( dLink );
    if ( MCAs0 != NULL )
      delete [] MCAs0;
    MCAs0 = new char [ XMAPBUFSIZE ];
    dLinkCount = 0;
    MCAsReady = false;  // MCAs のバッファに有効なデータが無い
    connect( dLink, SIGNAL( readyRead() ), this, SLOT( receiveMCAs() ),
             Qt::UniqueConnection );
    dLink->connectToHost( host, port );
  }
}
```

```
void AUnitXMAP::_setEnable( bool /*enable*/ )
{
  ConnectedToSSDServer = false;
  // 本当にこれでいいか stars との接続が on/off されると
  // かならず、裏でつながっているダイレクとのコネクションが切れたことにされる
}

double AUnitXMAP::SetTime( double dtime ) // in sec, この関数は、複数ステップ化できな
い
{
  IsBusy2On( Dev, "SetTime" );
  s->SendCMD2( Uid, Dev, "RunStop" );   // コマンド連続発行可能か? いちおういけてる
  s->SendCMD2( Uid, DevCh, "SetPresetValue", QString::number( dtime ) );
  setTime = dtime;

  return setTime;
}

bool AUnitXMAP::GetValue( void )
{
  IsBusy2On( Dev, "GetValue" );
  // 変則 : この IsBusy2 は @GetMCAs Ok: を受けても消さない
  //        data-link 経由で完全なデータをもらった時に消す
  //    s->SendCMD2( Uid, Dev, "GetValues" );    // new mcas
  s->SendCMD2( Uid, Dev, "GetMCAs" );

  return false;
}

bool AUnitXMAP::GetValue0( void )  // 値読み出しコマンドの前に何か必要なタイプの場合
{
  bool rv = false;

  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "GetValue0c0" );
    s->SendCMD2( Uid, Dev, "RunStop" );
    rv = true;
    LocalStage++;
    break;
  case 1:
    IsBusy2On( Dev, "GetValue0c1" );
    IsBusy = true;
    LastFunc = "GetValue0c1";
    emit ChangedIsBusy1( Dev );
    s->SendCMD2( Uid, Dev, "RunStart" );
    rv = false;
    LocalStage++;
    break;
  }

  return rv;
}

// 値読み出しコマンドの前に何か必要なタイプの場合
// 別バージョン、presetTime 等の終了条件無しにしてある
// 連続スキャン (差分で値を見る)モード用
bool AUnitXMAP::GetValue02( void )
{
  bool rv = false;

  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "GetValue0c0" );
    s->SendCMD2( Uid, Dev, "SetPresetType", "NONE" );
    rv = true;
```

```
    LocalStage++;
    break;
  case 1:
    IsBusy2On( Dev, "GetValue0c1" );
    s->SendCMD2( Uid, Dev, "RunStop" );
    rv = true;
    LocalStage++;
    break;
  case 2:
    IsBusy2On( Dev, "GetValue0c2" );
    IsBusy = true;
    LastFunc = "GetValue0c1";
    emit ChangedIsBusy1( Dev );
    s->SendCMD2( Uid, Dev, "RunStart" );
    rv = false;
    LocalStage++;
    break;
  }
  return rv;
}

/* 連続スキャン対応 */
// 連続スキャンの後にノーマルモードに戻す
bool AUnitXMAP::Close( void )
{
  IsBusy2On( Dev, "GetValue0c0" );
  s->SendCMD2( Uid, Dev, "RunStop" );

  return false;
}

void AUnitXMAP::RunStart( void )
{
  s->SendCMD2( Uid, Dev, "RunStart" );
}

void AUnitXMAP::RunStop( void )
{
  s->SendCMD2( Uid, Dev, "RunStop" );
}

void AUnitXMAP::RunResume( void )
{
  s->SendCMD2( Uid, Dev, "Resume" );
}

bool AUnitXMAP::GetMCAs( void )
{
  IsBusy2On( Dev2, "GetMCAs" );
  // 変則 : この IsBusy2 は @GetMCAs Ok: を受けても消さない
  //        data-link 経由で完全なデータをもらった時に消す
  s->SendCMD2( Uid, DevCh, QString( "GetMCAs" ) );

  return false;
}

void AUnitXMAP::getMCALength( SMsg msg )
{
  if ( msg.From() == Dev ) {  //   // Check !!!!! DevCh/Drv
    MCALength = msg.Val().toInt();
  }
}

bool AUnitXMAP::GetStat( void )
{
  bool rv = false;
  IsBusy2On( Dev, "GetStat" );
  s->SendCMD2( Uid, Dev, "GetStatistics" );

  return rv;
}

void AUnitXMAP::ReactGetStat( SMsg msg )
{
  if ( ( msg.From() == DevCh ) || ( msg.From() == Dev ) ) {  // Check !!!!! DevCh/Drv
    IsBusy2Off( Dev );
    MCAStats = msg.Vals();
  }
}

double AUnitXMAP::stat( int ch, STATELM i )
{
  double rv = 0;

  if ( MCAStats.count() >= 140 ) {
    rv = MCAStats.at( ch * 7 + (int)i ).toDouble();
  }

  return rv;
}

bool AUnitXMAP::SetRealTime( int ch, double val )
{
  bool rv = false;

  IsBusy2On( Dev, "SetRealTime2" );
  s->SendCMD2( Uid, Dev, "SetRealTime",
               QString::number( ch ) + " " + QString::number( val ) );

  return rv;
}

bool AUnitXMAP::GetRealTime( int ch )
{
  bool rv = false;

  IsBusy2On( Dev, "GetRealTime" );
  s->SendCMD2( Uid, Dev, "GetRealTime", QString::number( ch ) );

  return rv;
}

void AUnitXMAP::ReactGetRealTime( SMsg msg )
{
  int ch;

  if ( ( msg.From() == DevCh ) || ( msg.From() == Dev ) ) {  // Check !!!!! DevCh/Drv
    IsBusy2Off( Dev );
    MCARealTime[ ch = msg.Vals().at(0).toInt() ] = msg.Vals().at(1).toDouble();
    emit ReceivedNewMCARealTime( ch );
  }
}

double AUnitXMAP::realTime( int ch )
{
  return MCARealTime[ ch ];
}

#if 0
bool AUnitXMAP::SetLiveTime( double val )
{
```

```
  bool rv = false;

  if ( Type == "SSDP" ) {
    IsBusy2On( Dev, "SetLiveTime1" );
    s->SendCMD2( Uid, DevCh, "SetLiveTime", QString::number( val ) );
    rv = false;
  }

  return rv;
}
#endif

bool AUnitXMAP::SetLiveTime( int ch, double val )
{
  bool rv = false;

  IsBusy2On( Dev, "SetLiveTime2" );
  s->SendCMD2( Uid, Dev, "SetLiveTime",
               QString::number( ch ) + " " + QString::number( val ) );

  return rv;
}

bool AUnitXMAP::GetLiveTime( int ch )
{
  bool rv = false;

  IsBusy2On( Dev, "GetLiveTime2" );
  s->SendCMD2( Uid, Dev, "GetLiveTime", QString::number( ch ) );

  return rv;
}

void AUnitXMAP::ReactGetLiveTime( SMsg msg )
{
  int ch;

  if ( ( msg.From() == DevCh ) || ( msg.From() == Dev ) ) {  // Check !!!!! DevCh/Drv
    IsBusy2Off( Dev );
    MCALiveTime[ ch = msg.Vals().at(0).toInt() ] = msg.Vals().at(1).toDouble();
    emit ReceivedNewMCALiveTime( ch );
  }
}

double AUnitXMAP::liveTime( int ch )
{
  return MCALiveTime[ ch ];
}

void AUnitXMAP::SetLowLimit( int ch, int llpix )
{
  if ( ch < MaxSSDs ) {
    //    MCALowLimit[ ch ] = llpix;
    s->SendCMD2( "SSDSetting", Dev,
                 QString( "SetLLimit %1 %2" ).arg( ch ).arg( llpix ) );
  } else {
    qDebug() << "Setting LowLimit the ch " << ch << "is too big";
  }
}

void AUnitXMAP::ReactGetDataLinkCh( SMsg msg )
{
  if ( msg.From() == Dev ) {
    if ( msg.Vals().count() == 2 ) {
      IsBusy2Off( Dev );
      QString NewDataLinkHostName = msg.Vals().at(0);
      int NewDataLinkHostPort = msg.Vals().at(1).toInt();
      if ( ( ! ConnectedToSSDServer ) ||
           ( ( NewDataLinkHostName != DataLinkHostName )
             &&( NewDataLinkHostPort != DataLinkHostPort ) ) ) {
        DataLinkHostName = NewDataLinkHostName;
        DataLinkHostPort = NewDataLinkHostPort;
        ConnectedToSSDServer = true;
        ConnectToXMAPDataLinkServer( DataLinkHostName, DataLinkHostPort );
        qDebug() << "Connect to SSD server" << DataLinkHostName << DataLinkHostPort;
      }
    }
  }
}

void AUnitXMAP::setDark( void )
{
  Dark = Value.toDouble() / ( ( setTime != 0 ) ? setTime : 1 );
  DarkCountsInROI.clear();
  DarkCountsAll.clear();
  DarkTotalEvents.clear();
  for ( int i = 0; i < CountsInROI.count(); i++ ) {
    DarkCountsInROI << CountsInROI.at(i) / ( ( setTime != 0 ) ? setTime : 1 );
    DarkCountsAll << CountsAll.at(i) / ( ( setTime != 0 ) ? setTime : 1 );
    DarkTotalEvents << TotalEvents.at(i) / ( ( setTime != 0 ) ? setTime : 1 );
  }
  DarkICRs = ICRs;
  setDarkTime = setTime;
  emit newDark( Dark );
}


quint32 AUnitXMAP::getAMCAdata( int ch, int pixel )
{
  if ( !MCAsReady )
    return 0;
  return *((quint32 *)( MCAs + AXMAPBUF * ch + XMAPHEAD ) + pixel );
}

quint32 *AUnitXMAP::getAMCA( int ch )
{
  if ( !MCAsReady )
    return NULL;
  return (quint32 *)( MCAs + AXMAPBUF * ch + XMAPHEAD );
}

XMAPHead AUnitXMAP::getAMCAHead( int ch )
{
  XMAPHead rv;

  if ( !MCAsReady )
    return rv;
  rv.ch       = *(qint64*)( MCAs + AXMAPBUF * ch +  0 );
  rv.stat     = *(qint64*)( MCAs + AXMAPBUF * ch +  8 );
  rv.len      = *(qint64*)( MCAs + AXMAPBUF * ch + 16 );
  rv.realTime = *(double*)( MCAs + AXMAPBUF * ch + 24 );
  rv.liveTime = *(double*)( MCAs + AXMAPBUF * ch + 32 );
  rv.icr      = *(double*)( MCAs + AXMAPBUF * ch + 40 );
  return rv;
}

void AUnitXMAP::setGain( int ch, double gain )
{
  s->SendCMD2( Uid, Dev, QString( "SetPreAMPGain %1 %2" ).arg( ch ).arg( gain ) );
}

void AUnitXMAP::ReceiveValues( SMsg msg )
```

```
{
  QString buf;

  CountsInROI.clear();
  CountsAll.clear();
  TotalEvents.clear();
  ICRs.clear();

  if ( ( msg.From() == Dev ) && ( msg.Msgt() == GETVALUES ) ) { // Check !!!!! DevCh/
Drv
    int sum = 0;
    for ( int i = 0; i < MaxSSDs; i++ ) {
      if ( SSDUsingCh[i] ) {
        sum += msg.Vals().at( i + 1 ).toInt();
      }
    }
    Value = QString::number( sum );
    for ( int i = 0; i < MaxSSDs; i++ ) {
      CountsInROI << msg.Vals().at( i + 1 ).toInt();
      CountsAll   << msg.Vals().at( i + 1 + MaxSSDs ).toInt();
      TotalEvents << msg.Vals().at( i + 1 + MaxSSDs * 2 ).toInt();
      ICRs        << msg.Vals().at( i + 1 + MaxSSDs * 3 ).toDouble();
    }

    Values = msg.Vals();

    emit newValue( Value );
    IsBusy2Off( Dev );
  }
}

void AUnitXMAP::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == Dev )
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}

void AUnitXMAP::receiveMCAs( void )
{
  uint bytes0, bytes;

  bytes0 = dLink->bytesAvailable();
  // 今届いた分を全部読んでもバッファサイズより小さいなら
  if ( dLinkCount + bytes0 <= XMAPBUFSIZE )
    bytes = bytes0;                    // 全部読む
  else
    bytes = XMAPBUFSIZE - dLinkCount;   // 大きいなら、読める分だけ読む

  bytes = dLinkStream->readRawData( MCAs0 + dLinkCount, bytes );
  dLinkCount += bytes;

  if ( dLinkCount >= XMAPBUFSIZE ) {
    IsBusy2Off( Dev );
    dLinkCount = 0;
    if ( MCAs != NULL ) delete [] MCAs;
    MCAs = MCAs0;              // 読み込みが完成したバッファ(MCAs0)を
                               // 最新のデータが置かれたバッファ(MCAs)に移し
    MCAs0 = new char [ XMAPBUFSIZE ];
                               // MCAs0 は次のデータを受けるために新しくする
    MCAsReady = true;          // MCAs のバッファに有効なデータがある

    CountsInROI.clear();
    CountsAll.clear();
    TotalEvents.clear();
    ICRs.clear();

    quint64 sum = 0;
    quint64 countsAll, countsInROI;
    for ( int i = 0; i < MaxSSDs; i++ ) {
      quint32 *aMCA = getAMCA( i );
      countsAll = countsInROI = 0;
      for ( int j = 0; j < (int)MCALength; j++ ) {
        if ( ( j >= ROIStart[i].toInt() )&&( j <= ROIEnd[i].toInt() ) )
          countsInROI += aMCA[j];
        countsAll += aMCA[j];
      }
      CountsAll << countsAll;
      sum += countsInROI;
      CountsInROI << countsInROI;
    }

    Value = QString::number( sum );
    for ( int i = 0; i < MaxSSDs; i++ ) {
      TotalEvents << 0;
      ICRs        << getAMCAHead( i ).icr;
    }
    emit LogMsg( "emitted New MCAs" );
    emit NewMCAsAvailable( MCAs );
    emit newValue( Value );
  }
}
```

```
#ifndef AUNITXMAP2_H
#define AUNITXMAP2_H

#include "ASensor.h"
#include "AUnitXMAP.h"   // AUnitXMAP と AUnitXMAP2 は継承関係にない

class AUnitXMAP2 : public ASensor
{
  Q_OBJECT

 public:
  AUnitXMAP2( void ) {};
  void init0( void );

  double stat( STATELM i );
  bool SetRealTime( double val );
  double SetTime( double t );

 private slots:
  void getNewValue( QString v );   // only for SSD childlen
  void getNewDark( double d );     // only for SSD childlen

};

#endif
```

```
#include "AUnitXMAP2.h"

void AUnitXMAP2::init0( void )
{
  connect( s, SIGNAL( AnsSetPresetValue( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetROIs( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  s->SendCMD2( "Init", "System", "flgon", DevCh );
}

double AUnitXMAP2::stat( STATELM i )
{
  return ((AUnitXMAP*)TheParent)->stat( Ch.toInt(), i );
}

void AUnitXMAP2::getNewValue( QString )
{
  Value = QString::number( ((AUnitXMAP*)TheParent)->getCountsInROI().at( Ch.toInt() )
 );
}

void AUnitXMAP2::getNewDark( double )
{
  Dark = ((AUnitXMAP*)TheParent)->getDarkCountsInROI().at( Ch.toInt() );
}

double AUnitXMAP2::SetTime( double dtime ) // in sec, この関数は、複数ステップ化でき
ない
{
  IsBusy2On( Dev, "SetTime" );
  s->SendCMD2( Uid, Dev, "RunStop" );   // コマンド連続発行可能か? いちおういけてる
  s->SendCMD2( Uid, DevCh, "SetPresetValue", QString::number( dtime ) );
  setTime = dtime;

  return setTime;
}

bool AUnitXMAP2::SetRealTime( double val )
{
  bool rv = false;

  IsBusy2On( Dev, "SetRealTime1" );
  s->SendCMD2( Uid, DevCh, "SetRealTime", QString::number( val ) );
  rv = false;

  return rv;
}
```

```
#ifndef AUNITPAM_H
#define AUNITPAM_H

#include "ASensor.h"

class AUnitPAM : public ASensor
{
  Q_OBJECT

 public:
  AUnitPAM( void ) {};

  void init0( void );
  virtual void init00( void );
  bool InitSensor( void );

  bool isAutoRangeAvailable( void ) { return true; };
  bool GetValue( void );
  void SetRange( int range );
  virtual void _SetRange( int range );
  virtual bool _GetValue( void );
  double SetTime( double dtime );
  virtual void _SetTime( double t );
};

class AUnitPAM2 : public AUnitPAM
{
  Q_OBJECT

 public:
  AUnitPAM2( void ) {};
  void init00( void );

  void _SetRange( int range );
  bool _GetValue( void );
  void _SetTime( double t );

 private slots:
  void RcvAnsGetValueOfDriver( SMsg msg );
};

#endif
```

```
#include "AUnitPAM.h"

void AUnitPAM::init0( void )
{
  connect( s, SIGNAL( AnsRead( SMsg ) ), this, SLOT( SetCurPos( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsReset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetAutoRange( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetDataFormat( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetZeroCheck( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetNPLCycles( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );

  init00();
}

void AUnitPAM::init00( void )
{
  s->SendCMD2( "Init", DevCh, "IsBusy" );
}

void AUnitPAM2::init00( void )    // PAM と PAM2 で違ってる
{
  connect( s, SIGNAL( AnsRead( SMsg ) ),this, SLOT( RcvAnsGetValueOfDriver( SMsg ) ),
           Qt::UniqueConnection );

  s->SendCMD2( "Init", Dev, "IsBusy" );
}

bool AUnitPAM::InitSensor( void )
{
  bool rv = false;

  QString dev;
  if ( Type == "PAM" ) { dev = Dev; } else { dev = Dev; } // !!
  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "InitSensor-c0" );
    s->SendCMD2( "Scan", dev, "Reset", "" );
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev, "InitSensor-c1" );
    if ( Type == "PAM" )
      s->SendCMD2( "Scan", dev, "SetAutoRangeEnable", "1" );
    if ( Type == "PAM2" )
      s->SendCMD2( "Scan", dev, "SetAutoRangeEnable " + Ch, "1" );
    LocalStage++;
    rv = true;
    break;
  case 2:
    IsBusy2On( Dev, "InitSensor-c2" );
    if ( Type == "PAM" )
      s->SendCMD2( "Scan", dev, "SetDataFormatElements", "READ" );
    if ( Type == "PAM2" )
      s->SendCMD2( "Scan", dev, "SetDataFormatElements", "CURR1,CURR2" );

    if ( Type == "PAM" ) {
      LocalStage++;
      rv = true;
```

```
    } else {
      rv = false;      // PAM2 は ZeroCheck の設定ないらしい
      LocalStage = 4;
    }
    break;
  case 3:
    IsBusy2On( Dev, "InitSensor-c3" );
    s->SendCMD2( "Scan", dev, "SetZeroCheckEnable", "0" );
    rv = false;
    LocalStage++;
    break;
  default:
    rv = false;
  }
  return rv;
}

bool AUnitPAM::GetValue( void )
{
  return _GetValue();
}

bool AUnitPAM::_GetValue( void )
{
  IsBusy2On( Dev, "GetValue" );
  s->SendCMD2( Uid, DevCh, "Read" );

  return false;
}

bool AUnitPAM2::_GetValue( void )
{
  IsBusy2On( Dev, "GetValue" );
  s->SendCMD2( Uid, Dev, "Read" );

  return false;
}

void AUnitPAM2::RcvAnsGetValueOfDriver( SMsg msg )  // driver 名だけで呼ばれる場合
{
  if ( ( msg.From() == Dev ) && ( msg.Msgt() == READ ) ) {
    Values = msg.Val().split( QChar( ',' ) );
    Value = Values.at( Ch.toInt() ); // 親ドライバ宛の返答から自分用の答えを選り分け
る
    emit newValue( Value );
    IsBusy2Off( Dev );
  }
}

double AUnitPAM::SetTime( double dtime ) // in sec// この関数は、複数ステップ化できな
い
{
  double time;

  IsBusy2On( Dev, "SetTime" );
  // 1 sec -> 1/60 sec
  time = dtime * 60;
  if ( time < 1 ) time = 1;
  if ( time > 40 ) time = 40;

  _SetTime( time );

  setTime = time / 60;    // これで「秒」単位の普通の時間に戻ってる
  return setTime;
}
```

```
void AUnitPAM::_SetTime( double time )
{
  s->SendCMD2( Uid, DevCh, "SetNPLCycles", QString::number( time ) );
}

void AUnitPAM2::_SetTime( double time )
{
  s->SendCMD2( Uid, Dev, "SetNPLCycles " + Ch, QString::number( time ) );
}

void AUnitPAM::SetRange( int range )
{
  IsBusy2On( Dev2, "SetRange" );
  _SetRange( range );
  SelectedRange = range;
}

void AUnitPAM::_SetRange( int range )
{
  s->SendCMD2( "Scan", DevCh, "SetRange", QString( "2E%1" ).arg( range ) );
}

void AUnitPAM2::_SetRange( int range )
{
  s->SendCMD2( "Scan", Dev, "SetRange " + Ch, QString( "2E%1" ).arg( range ) );
}
```

```
#ifndef AUNITOTC_H
#define AUNITOTC_H

#include "ASensor.h"

class AUnitOTC : public ASensor
{
  Q_OBJECT

 public:
  AUnitOTC( void );
  void init0( void );
  virtual void init00( void );
  bool InitSensor( void );
  virtual bool _InitSensor( void );

  bool GetValue0( void );
  void AskIsBusy( void );
  virtual void _AskIsBusy( void );
  double SetTime( double dtime );

 private slots:
  void SetIsBusyByMsg( SMsg msg );
};

class AUnitOTC2 : public AUnitOTC
{
  Q_OBJECT

 public:
  AUnitOTC2( void );
  void init00( void );
  bool _InitSensor( void );

  void _AskIsBusy( void );
  bool isAutoRangeAvailable( void ) { return true; };
  bool GetRange( void );
  void SetRange( int range );

 private slots:
  void ReactGetRange( SMsg msg );
};




#endif
```

```
#include "AUnitOTC.h"

AUnitOTC::AUnitOTC( void )
{
}

AUnitOTC2::AUnitOTC2( void )
{
}

void AUnitOTC::init0( void )
{
  connect( s, SIGNAL( AnsReset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetMode( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetCountPreset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsCounterReset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
             Qt::UniqueConnection );
  connect( s, SIGNAL( AnsRun( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsStop( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );

  s->SendCMD2( "Init", Dev, "IsBusy" );
  s->SendCMD2( "Init", Dev, "Reset" );
  s->SendCMD2( "Init", Dev, "SetMode", "0" );

  init00();
}

void AUnitOTC::init00( void )
{
}

void AUnitOTC2::init00( void )
{
  connect( s, SIGNAL( AnsReset( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetAutoRange( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetZeroCheck( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsSetRange( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGetRange( SMsg ) ), this, SLOT( ReactGetRange( SMsg ) ),
           Qt::UniqueConnection );
}

bool AUnitOTC::InitSensor( void )
{
  return _InitSensor();
}

bool AUnitOTC::_InitSensor( void )
{
  return false;
}

bool AUnitOTC2::_InitSensor( void )
{
  if ( The2ndDev == NULL ) {
    qDebug() << "InitSensor:: The2ndDev is not initialized" << Uid << Name;
    return false;
```

```
  }

  bool rv = false;
  // CNT2, OTC2 のとき カウンタの向こうにつながるのは
  // keithley なのでそれ用の処理をしておく
  QString Type2 = The2ndDev->type();
  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev2, "InitSensor-c0" );
    s->SendCMD2( "Scan", DevCh2, "Reset", "" );
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev2, "InitSensor-c1" );
    if ( autoRange ) {
      if ( Type2 == "PAM" )
        s->SendCMD2( "Scan", DevCh2, "SetAutoRangeEnable", "1" );
      if ( Type2 == "PAM2" )
        s->SendCMD2( "Scan", Dev2, "SetAutoRangeEnable " + Ch2, "1" );
      LocalStage = 3;
    } else {
      if ( Type2 == "PAM" )
        s->SendCMD2( "Scan", DevCh2, "SetAutoRangeEnable", "0" );
      if ( Type2 == "PAM2" )
        s->SendCMD2( "Scan", Dev2, "SetAutoRangeEnable " + Ch2, "0" );
      LocalStage = 2;
    }
    rv = true;
    break;
  case 2:
    IsBusy2On( Dev2, "InitSensor-c2" );
    if ( Type2 == "PAM" ) {
      s->SendCMD2( "Scan", DevCh2, "SetRange", QString( "2E%1" ).arg( SelectedRange )
 );
      LocalStage++;
      rv = true;
    }
    if ( Type2 == "PAM2" ) {
      s->SendCMD2( "Scan", Dev2, "SetRange " + Ch2,
                   QString( "2E%1" ).arg( SelectedRange ) );
      LocalStage+=2;     // PAM2 の時は、LocalStage == 3 をとばす
      rv = false;
    }
    break;
  case 3:
    IsBusy2On( Dev2, "InitSensor-c3" );
    s->SendCMD2( "Scan", DevCh2, "SetZeroCheckEnable", "0" );
    rv = false;
    LocalStage++;
    break;
  default:
    rv = false;
  }

  return rv;
}



void AUnitOTC::AskIsBusy( void )
{
  _AskIsBusy();
}

void AUnitOTC::_AskIsBusy( void )
```

```
{
  s->SendCMD2( Uid, DevCh, "IsBusy" );
}

void AUnitOTC2::_AskIsBusy( void )
{
}

void AUnitOTC::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == Dev )   // Check !!!!! DevCh/Drv
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}

bool AUnitOTC::GetValue0( void )  // 値読み出しコマンドの前に何か必要なタイプの場合
{
  bool rv = false;

  switch( LocalStage ) {
  case 0:
    IsBusy2On( Dev, "GetValue0c0" );
    s->SendCMD2( Uid, Dev, "CounterReset" );
    LocalStage++;
    rv = true;
    break;
  case 1:
    IsBusy2On( Dev, "GetValue0c1" );
    IsBusy = true;
    LastFunc = "GetValue0c1";
    emit ChangedIsBusy1( Dev );
    s->SendCMD2( Uid, Dev, "Run" );
    LocalStage++;
    rv = false;
    break;
  }

  return rv;
}

bool AUnitOTC2::GetRange( void ) // CNT2, OTC2
{
  QString Type2 = The2ndDev->type();
  IsBusy2On( Dev2, "GetRange" );
  if ( Type2 == "PAM" )
    s->SendCMD2( Uid, DevCh2, QString( "GetRange" ) );
  if ( Type2 == "PAM2" )
    s->SendCMD2( Uid, Dev2, QString( "GetRange " ) + Ch2 );

  return false;
}

void AUnitOTC2::ReactGetRange( SMsg msg )  // CNT2, OTC2
{
  double range = RangeL;
  if ( ( msg.From() == DevCh2 ) || ( msg.From() == Dev2 ) ) {
    QString Type2 = The2ndDev->type();
    if ( Type2 == "PAM" ) {
      range = log10( msg.Vals().at(0).toDouble() / 2.1 );
    }
```

```
    if ( Type2 == "PAM2" ) {
      if ( msg.Vals().at(0).toInt() == Ch2.toInt() ) {
        range = log10( msg.Vals().at(1).toDouble() / 2.1 );
      } else {
        return;
      }
    }

    IsBusy2Off( Dev2 );
    if ( range > RangeU ) range = RangeU;
    if ( range < RangeL ) range = RangeL;
    emit AskedNowRange( (int)range );
  }
}

double AUnitOTC::SetTime( double dtime ) // in sec // この関数は、複数ステップ化でき
ない
{
  int M, N;

  IsBusy2On( Dev, "SetTime" );
  N = log10( dtime * 10 );
  M = ceil( dtime / pow( 10., N - 1 ) );
  s->SendCMD2( Uid, Dev, "SetCountPreset", QString( "%1,%2" ).arg( M ).arg( N ) );
  setTime = M * pow( 10, N ) * 0.1;  // これで秒単位の普通の時間に戻ってる

  return setTime;
}

void AUnitOTC2::SetRange( int range )
{
  if ( The2ndDev == NULL ) {
    qDebug() << "SetRange : The2ndDev is not initialized" << Uid << Name;
    return;
  }

  IsBusy2On( Dev2, "SetRange" );
  // CNT2, OTC2 のとき カウンタの向こうにつながるのは
  // keithley ( PAM/PAM2 )なのでそれ用の処理をしておく
  QString Type2 = The2ndDev->type();
  if ( Type2 == "PAM" ) {
    s->SendCMD2( "Scan", DevCh2, "SetRange", QString( "2E%1" ).arg( range ) );
  }
  if ( Type2 == "PAM2" ) {
    s->SendCMD2( "Scan", Dev2, "SetRange " + Ch2,
                 QString( "2E%1" ).arg( range ) );
  }
  SelectedRange = range;
}
```

```
#ifndef AUNITENC_H
#define AUNITENC_H

#include "ASensor.h"

class AUnitENC : public ASensor
{
  Q_OBJECT

 public:
  AUnitENC( void );

  void init0( void );
  virtual void init00( void );
  bool InitSensor( void );
  virtual bool _InitSensor( void );

  void SetValue( double v );
};

class AUnitENC2 :public AUnitENC
{
  Q_OBJECT

 public:
  AUnitENC2( void );
  void init00( void );
  bool _InitSensor( void );

  double SetTime( double dtime );
  void AskIsBusy( void );
  bool QStart( void );
  bool QRead( void );

 private slots:
  void SetIsBusyByMsg( SMsg msg );
  void RcvQGetData( SMsg msg );
  void RcvStat( SMsg msg );
};

#endif
```

```cpp
#include "AUnitENC.h"

AUnitENC::AUnitENC( void )
{
}

AUnitENC2::AUnitENC2( void )
{
}

void AUnitENC::init0( void  )
{
  init00();
}

void AUnitENC::init00( void )
{
  connect( s, SIGNAL( EvChangedValue( SMsg ) ), this, SLOT( SetCurPos( SMsg ) ),
           Qt::UniqueConnection );
  s->SendCMD2( "Init", DevCh, "IsBusy" );
  s->SendCMD2( "Init", DevCh, "GetValue" );
}

void AUnitENC2::init00( void )
{
  connect( s, SIGNAL(AnsTrigger( SMsg )), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsStandBy( SMsg )), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsGetStat( SMsg )), this, SLOT( RcvStat( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL(AnsGetData( SMsg )), this, SLOT( RcvQGetData( SMsg ) ),
           Qt::UniqueConnection );
}

bool AUnitENC::InitSensor( void )
{
  return _InitSensor();
}

bool AUnitENC::_InitSensor( void )
{
  return false;
}

bool AUnitENC2::_InitSensor( void )
{
  bool rv = false;

  if ( Type == "ENC2" ) {
    IsBusy2On( Dev, "InitSensor-c0" );
    s->SendCMD2( "Init", DevCh, "GetValue" );
    LocalStage++;
    rv = false;
  }

  return rv;
}

void AUnitENC::SetValue( double v )
{
  s->SendCMD2( Uid, DevCh, "SetValue", QString::number( DLastSetV = v ) );
}

void AUnitENC2::AskIsBusy( void )
```

```cpp
{
  s->SendCMD2( Uid, DevCh, "IsBusy" );
}

void AUnitENC2::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == DevCh )
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}

bool AUnitENC2::QStart( void )
{
  IsBusy2On( Dev, "Start" );
  s->SendCMD2( Uid, DevCh, "StandBy" );
  return false;
}

bool AUnitENC2::QRead( void )
{
  IsBusy2On( Dev, "Read" );
  s->SendCMD2( Uid, DevCh, "GetData" );

  return false;
}

double AUnitENC2::SetTime( double dtime ) // in sec // この関数は、複数ステップ化でき
ない
{
  setTime = dtime;          // setTime できたと見せかけるだけ。

  return setTime;
}

void AUnitENC2::RcvQGetData( SMsg msg )
{
  if ( ( ( msg.From() == DevCh )||( msg.From() == Dev ) )
       && ( ( msg.Msgt() == GETDATAPOINTS )
            || ( msg.Msgt() == QGETDATA )
            || ( msg.Msgt() == GETDATA ) ) ) {

    Values = msg.Vals();
    emit newQData();
    IsBusy2Off( Dev );
  }
}


void AUnitENC2::RcvStat( SMsg msg )
{
  if ( ( ( msg.From() == DevCh )||( msg.From() == Dev ) )  // Check !!!!! DevCh/Drv
       && ( ( msg.Msgt() == GETSTAT ) ) ) {
    //      Values = msg.Vals();
    //      emit newQData();
    IsBusy2Off( Dev );
  }
}
```

```
#ifndef AUNITAIO_H
#define AUNITAIO_H

#include "ASensor.h"
#include "AMotor.h"

class AUnitAIOi : public ASensor
{
  Q_OBJECT

 public:
  AUnitAIOi( void );
  void init0( void );
  double SetTime( double dtime );

 private slots:
  void SetIsBusyByMsg( SMsg msg );
};

class AUnitAIOo : public AMotor
{
  Q_OBJECT

  double MaxV, MinV;

 public:
  AUnitAIOo( void );
  void init0( void );

  void setMaxV( QString maxv ) { MaxV = maxv.toDouble(); };
  void setMinV( QString minv ) { MinV = minv.toDouble(); };

  void CloseShutter( bool close );
  void GoMaxAbs( double start, double end, int steps );
  void GoMaxAbsQ( double start, double end, int steps, double time );
  void GoMaxRel( double width, int steps );
  void GoMaxRelQ( double width, int steps, double time );

 private slots:
  void SetIsBusyByMsg( SMsg msg );
};

#endif
```

```
#include "AUnitAIO.h"

AUnitAIOi::AUnitAIOi( void )
{
}

AUnitAIOo::AUnitAIOo( void )
{
  MaxV = 0;
  MinV = 0;
}

void AUnitAIOi::init0( void )
{
  s->SendCMD2( "Init", Dev, "IsBusy" );
}

void AUnitAIOo::init0( void )
{
  connect( s, SIGNAL( AnsGoMaxAbs( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsGoMaxRel( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );
  connect( s, SIGNAL( AnsShutterOff( SMsg ) ), this, SLOT( ClrBusy( SMsg ) ),
           Qt::UniqueConnection );

  s->SendCMD2( "Init", DevCh, "Init" );
  s->SendCMD2( "Init", Dev, "IsBusy" );
}

void AUnitAIOi::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == Dev )   // Check !!!!! DevCh/Drv
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}

void AUnitAIOo::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == Dev )   // Check !!!!! DevCh/Drv
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}

double AUnitAIOi::SetTime( double dtime ) // in sec // この関数は、複数ステップ化でき
ない
{
  setTime = dtime;          // setTime できたと見せかけるだけ。

  return setTime;
}

/** MStab *************************************************************************/
```

```
void AUnitAIOo::CloseShutter( bool close )
{
  if ( close ) {
    s->SendCMD2( Uid, Dev, "ShutterOff 1" );
  } else {
    s->SendCMD2( Uid, Dev, "ShutterOff 0" );
  }
}

void AUnitAIOo::GoMaxAbs( double start, double end, int steps )
{
  IsBusy2On( Dev, "GoMaxAbs" );
  s->SendCMD2( Uid, Dev,
               QString( "GoMaxAbs 0 %1 %2 %3" ).arg( start ).arg( end ).arg( steps )
);
}

void AUnitAIOo::GoMaxAbsQ( double start, double end, int steps, double time )
{
  if ( Type == "AIOo" ) {
    IsBusy2On( Dev, "GoMaxAbsQ" );
    s->SendCMD2( Uid, Dev,
                 QString( "GoMaxAbs 1 %1 %2 %3 %4" )
                 .arg( start ).arg( end ).arg( steps ).arg( time ) );
  }
}

void AUnitAIOo::GoMaxRel( double width, int steps )
{
  if ( Type == "AIOo" ) {
    IsBusy2On( Dev, "GoMaxRel" );
    s->SendCMD2( Uid, Dev,
                 QString( "GoMaxRel 0 %1 %2" ).arg( width ).arg( steps ) );
  }
}

void AUnitAIOo::GoMaxRelQ( double width, int steps, double time )
{
  if ( Type == "AIOo" ) {
    IsBusy2On( Dev, "GoMaxRelQ" );
    s->SendCMD2( Uid, Dev,
                 QString( "GoMaxRel 1 %1 %2 %3" )
                 .arg( width ).arg( steps ).arg( time ) );
  }
}
```

```
#ifndef AUNITLSR_H
#define AUNITLSR_H

#include "ASensor.h"

class AUnitLSR : public ASensor
{
  Q_OBJECT

 public:
  AUnitLSR( void );
  void init0( void );

 private slots:
  void OnReportCurrent( SMsg msg );
  void OnReportInjection( SMsg msg );

 signals:
  void NewRingCurrent( QString val, QStringList vals );
  void NewInjectionReport( QString val, QStringList vals );
};

#endif
```

```
#include "AUnitLSR.h"

AUnitLSR::AUnitLSR( void )
{
}

void AUnitLSR::init0( void )
{
  connect( s, SIGNAL(EvReportCurrent( SMsg )), this, SLOT(OnReportCurrent( SMsg )),
           Qt::UniqueConnection );
  connect( s, SIGNAL(EvReportInjection( SMsg )), this, SLOT(OnReportInjection( SMsg )
),
           Qt::UniqueConnection );
  s->SendCMD2( "Init", Dev, "flgon", Ch );
}

void AUnitLSR::OnReportCurrent( SMsg msg )
{
  if ( msg.From() == DevCh ) {
    Values = msg.Val().simplified().split( QRegExp( "\\s" ) );
    LastValue = Value;
    Value = Values[ Values.count() - 1 ];
    emit NewRingCurrent( Value, Values );
  }
}

void AUnitLSR::OnReportInjection( SMsg msg )
{
  if ( msg.From() == DevCh ) {
    Values = msg.Val().simplified().split( QRegExp( "\\s" ) );
    LastValue = Value;
    Value = Values[ Values.count() - 1 ];
    emit NewInjectionReport( Value, Values );
    emit newValue( Value );
  }
}
```

```
#ifndef AUNITCCG_H
#define AUNITCCG_H

#include "ASensor.h"

class AUnitCCG : public ASensor
{
  Q_OBJECT

 public:
  AUnitCCG( void );

  void init0( void );

 private slots:
  void SetCurPos( SMsg msg );
};

#endif
```

```
#include "AUnitCCG.h"

AUnitCCG::AUnitCCG( void )
{
}

void AUnitCCG::init0( void )
{
  s->SendCMD2( "Init", DevCh, "IsBusy" );
}

void AUnitCCG::SetCurPos( SMsg msg )
{
  QString buf;

  if ( ( msg.From() == DevCh )
       && ( ( msg.Msgt() == GETVALUE ) || ( msg.Msgt() == EvCHANGEDVALUE )
            || ( msg.Msgt() == READ ) ) ) {
    Value = msg.Vals().at(0);
    emit newValue( Value );
    IsBusy2Off( Dev );
  }
}
```

```
#ifndef AUNITFP23_H
#define AUNITFP23_H

#include "ASensor.h"

class AUnitFP23 : public ASensor
{
  Q_OBJECT

 public:
  AUnitFP23( void );
  void init0( void );

 private slots:
   void OnReportValue( SMsg msg );

 signals:
  void NewFP23Temperature( QString val );

};

#endif
```

```
#include "AUnitFP23.h"

AUnitFP23::AUnitFP23( void )
{
}

void AUnitFP23::init0( void )
{
  s->SendCMD2( "Init", Dev, "flgon", Ch );
  s->SendCMD2( "Init", "System", "flgon", Dev );

  connect( s, SIGNAL( EvReportValue(SMsg) ), this, SLOT( OnReportValue(SMsg) ),
           Qt::UniqueConnection );
  s->SendCMD2( "Init", DevCh, "IsBusy" );
}

void AUnitFP23::OnReportValue( SMsg msg )
{
  if ( msg.From() == DevCh ) {
    Values = msg.Val().simplified().split( QRegExp( "\\s" ) );
    LastValue = Value;
    Value = Values[ Values.count() - 1 ];
    emit NewFP23Temperature( Value );
  }
}
```

```
#ifndef AUNITEPIC_H
#define AUNITEPIC_H

#include "ASensor.h"

class AUnitEPIC : public ASensor
{
  Q_OBJECT

 public:
  AUnitEPIC( void );
  void init0( void );

  double SetTime( double dtime );

  private slots:
  void SetIsBusyByMsg( SMsg msg );
};

#endif
```

```
#include "AUnitEPIC.h"

AUnitEPIC::AUnitEPIC( void )
{
}

void AUnitEPIC::init0( void )
{
  s->SendCMD2( "Init", DevCh, "IsBusy" );
}

void AUnitEPIC::SetIsBusyByMsg( SMsg msg )
{
  if ( ( msg.From() == DevCh )
       && ( ( msg.Msgt() == ISBUSY ) || ( msg.Msgt() == EvISBUSY ) ) ) {
    IsBusy = ( msg.Val().toInt() == 1 );
    if ( IsBusy )
      LastFunc = "SetIsBusyByMsg";
    else
      LastFunc = "";
    emit ChangedIsBusy1( Dev );
  }
}

double AUnitEPIC::SetTime( double dtime ) // in sec // この関数は、複数ステップ化でき
ない
{
  setTime = dtime;          // setTime できたと見せかけるだけ。

  return setTime;
}
```