T1dBladeSetupDlg.cpp

T1dImpeller.cpp

```
if ( (_pVane->haveBaseClass("T1dImpeller") && !_pVane->getUseGeomCurve()) && (condition_Fan || isCompressor || isPump))
{
  QStringList _allbladeType;
  if (is3DVanelFan)
    _allbladeType = QStringList() << tr("Ruled free") << tr("Ruled");
  else  if (isCompressor)
    _allbladeType = QStringList() << tr("Ruled") << tr("Ruled free") << tr("Full 3D") << tr("Blade editor");
  else  if (isPump)
    _allbladeType = QStringList() << tr("Ruled free") << tr("Full 3D") << tr("Axial element free");
  else
    _allbladeType = QStringList() << tr("SCA") << tr("Flat") << tr("DCA") << tr("Ruled free") << tr("SCA_Flat");

  _bladeType = getCurrentBladeTypeName();
```

1

```
  // 3. set
  if (blade->bladeType == T1dBlade::Ruled_3D_Type)
    setHasMeanlineGeometry(false);
  if (blade->bladeType == T1dBlade::Ruled_3D_Free_Type)
    setHasMeanlineGeometry(false);
  if (blade->bladeType == T1dBlade::Free_3D_Type)
    setHasMeanlineGeometry(true);
  if (blade->bladeType == T1dBlade::Axial_Element_Free_Type)
    setHasMeanlineGeometry(true);
```

2

```
  else if (blade->bladeType == blade->bladeTypes::Axial_Element_Free_Type) // Axial_Element_Free_Type
  {
    PL << "beta1b" << "beta2b" << "wrapAngle" << "thick1" << "thick2" << "thickMax" << "thickmax_loc_chord";
  }
```

3

```
int T1dBladeSetupDlg::getCurrentBladeTypeName()
{
  int bladeTypeIndex = 0;
  if (isCompressor) { ... }
  else if (isPump)
  {
    if (_pVane->blade->bladeType == _pVane->blade->bladeTypes::Ruled_3D_Free_Type)
      bladeTypeIndex = 0;
    if (_pVane->blade->bladeType == _pVane->blade->bladeTypes::Free_3D_Type)
      bladeTypeIndex = 1;
    if (_pVane->blade->bladeType == _pVane->blade->bladeTypes::Axial_Element_Free_Type)
      bladeTypeIndex = 2;
  }
```

4

T1dBladeSetupDlg.cpp

T1dImpeller.cpp

```
void T1dBladeSetupDlg::bladeTypeChanged()
{
 if (isCompressor) { ... }
 else if (isPump)
 {
  switch (_bladeType)
  {
  case 0: _pVane->blade->bladeType = _pVane->blade->bladeTypes::Ruled_3D_Free_Type;
   break;
  case 1: _pVane->blade->bladeType = _pVane->blade->bladeTypes::Free_3D_Type;
   break;
  case 2: _pVane->blade->bladeType = _pVane->blade->bladeTypes::Axial_Element_Free_Type;
   break;
  }
 }

 // For Axial Element free Blade
 if (isPump && _pVane->haveBaseClass("T1dImpeller"))
 {
  foreach(QString s, propList)
  {
   bool isthink = (s == "thick1" || s == "thick2" || s == "thickMax" || s == "thickmax_loc_chord");
   foreach(TObject * o, objects)
   {
    if (property_t* p = o->property(s))
    {
     if ((o == _hub || o == _tip) && _bladeType == 2 && !isthink)
       SET_PROPERTY_READONLY(p);
     else
       SET_PROPERTY_WRITABLE(p);
    }
   }
  }
 }

else if (blade->bladeType == blade->bladeTypes::Axial_Element_Free_Type)
{
 UpdateBetaAxialElementFreeBlade();

}
return;
```

5

6

7

T1dImpeller.h

T1dImpeller.cpp

```
// by Feihong for axial element free blade
inline double getbetam(double betar, double phi);
inline double getbetar(double betam, double phi);
void getMeanbetab(double beta1b, double beta2b, double& beta1br, double& beta2br);
void getAllbetab(double beta1br, double beta2br);
void UpdateBetaAxialElementFreeBlade();
void Get_MbarTheta_conformalMapping_all(double beta1br, double beta2br, double wrapAngle);
double cal_Mbar_conformalMapping(TNurbsCurve* pZRCurve1, TNurbsCurve* pZRCurve2, double u1, double u2);
void get_mBetafromMbarTheta(TNurbsCurve* pZRCurve, double u1, double u2, TNurbsCurve* pmBetaCurve, TNurbsCurve* pMbarThetaCurve);
  else if (blade->bladeType == blade->bladeTypes::Axial_Element_Free_Type)
  {
    UpdateBetaAxialElementFreeBlade();
  }
  return;
}

 // by Feihong for axial element free blade
double T1dImpeller::getbetam(double betar, double phi) { ... }

double T1dImpeller::getbetar(double betam, double phi) { ... }

void T1dImpeller::getMeanbetab(double beta1b, double beta2b, double& beta1br, double& beta2br) { ... }

void T1dImpeller::getAllbetab(double beta1br, double beta2br) { ... }

void T1dImpeller::UpdateBetaAxialElementFreeBlade()
{
  // 0.
  auto beta1b = blade->getMeanSection()->beta1b;
  auto beta2b = blade->getMeanSection()->beta2b;
  auto wrapAngle = blade->getMeanSection()->wrapAngle;
  // 1.
  auto beta1br = 0., beta2br = 0.;
  getMeanbetab(beta1b, beta2b, beta1br, beta2br);
  // 2.
  Get_MbarTheta_conformalMapping_all(beta1br, beta2br, wrapAngle);
  // 3.
  getAllbetab(beta1br, beta2br);
}
```

T1dImpeller.cpp

```cpp
double T1dImpeller::getbetam(double betar, double phi)
{
  auto item_temp = sqrt(1 + 1 / (pow(tan(phi), 2)));
  auto tan_betam = tan(betar) / item_temp;
  return atan(tan_betam);
}

double T1dImpeller::getbetar(double betam, double phi)
{
  auto item_temp = sqrt(1 + 1 / (pow(tan(phi), 2)));
  auto tan_betar = tan(betam) * item_temp;
  return atan(tan_betar);
}

void T1dImpeller::getMeanbetab(double beta1b, double beta2b, double& beta1br, double& beta2br)
{
  // 1.mean
  Double2 TM_le, TM_te;
  pMeanContour->getTangentialDirection(ule_mean, TM_le);
  pMeanContour->getTangentialDirection(ute_mean, TM_te);
  beta1br = getbetar(beta1b, TM_le.angle());
  beta2br = getbetar(beta2b, TM_te.angle());
}

void T1dImpeller::getAllbetab(double beta1br, double beta2br)
{
  // 2.tip
  Double2 TT_le, TT_te;
  pShroudContour->getTangentialDirection(ule_shroud, TT_le);
  pShroudContour->getTangentialDirection(ute_shroud, TT_te);
  blade->tipSection->beta1b = getbetam(beta1br, TT_le.angle());
  blade->tipSection->beta2b = getbetam(beta2br, TT_te.angle());

  // 3.hub
  Double2 TH_le, TH_te;
  pHubContour1->getTangentialDirection(ule_hub, TH_le);
  pHubContour1->getTangentialDirection(ute_hub, TH_te);
  blade->hubSection->beta1b = getbetam(beta1br, TH_le.angle());
  blade->hubSection->beta2b = getbetam(beta2br, TH_te.angle());
}
```

T1dImpeller.cpp

```cpp
void T1dImpeller::Get_MbarTheta_conformalMapping_all(double beta1br, double beta2br, double wrapAngle)
{
  auto error = 0;
  // 1. mean (set Mbar-Theta distribution)
  // 1.1 calculate Start- and Endpoint
  auto Mbar_start = 0.;
  auto Mbar_end = cal_Mbar_conformalMapping(pMeanContour, pMeanContour, ule_mean, ute_mean);
  auto Theta_start = 0.;
  auto Theta_end = wrapAngle;
  Double2 pt_start = { Mbar_start, Theta_start };
  Double2 pt_end = { Mbar_end, Theta_end };

  // 1.2 drawing Theta-Mbar-diagram(pMbarThetaCurve)
  QVector<Double2> Mbar_Theta;
  TNurbsCurve* pMbarThetaCurve = new TNurbsCurve;
  get_MThetaCurve(pt_start, pt_end, tan(beta1br), tan(beta2br), Mbar_Theta, pMbarThetaCurve);

  // 1.3 tip
  // 1.3.1 u_tip
  auto Mbar_start_tip = cal_Mbar_conformalMapping(pMeanContour, pShroudContour, ule_mean, ule_shroud);
  auto fo = false;
  auto ut = pMbarThetaCurve->getUfromX(Mbar_start_tip, fo);
  // 1.3.2 Tip_StartPoint
  double Theta_start_tip;
  pMbarThetaCurve->getPoint(ut, Mbar_start_tip, Theta_start_tip);
  blade->tipSection->wrapAngle = wrapAngle - Theta_start_tip;

  // 1.4 hub_StartPoint
  auto Mbar_start_hub = cal_Mbar_conformalMapping(pMeanContour, pHubContour1, ule_mean, ule_hub);
  auto Theta_start_hub = Mbar_start_hub / tan(beta1br);
  blade->hubSection->wrapAngle = wrapAngle - Theta_start_hub;

  // 2. transform from Mbar-theta to m-beta
  // 2.1 Mean
  get_mBetafromMbarTheta(pMeanContour, ule_mean, ute_mean, pMeanBeta, pMbarThetaCurve);
  // 2.2 Tip
  get_mBetafromMbarTheta(pShroudContour, ule_shroud, ute_shroud, pShroudBeta, pMbarThetaCurve);
  // 2.3 Hub
  get_mBetafromMbarTheta(pHubContour1, ule_hub, ute_hub, pHubBeta, pMbarThetaCurve);
}
```

```cpp
void T1dImpeller::get_mBetafromMbarTheta(TNurbsCurve* pZRCurve, double u1, double u2, TNurbsCurve* pmBetaCurve, TNurbsCurve* pMbarThetaCurve)
{
  // 1.getPoint from MbarTheta
  auto Mbar_start = 0., theta_start = 0.;
  pMbarThetaCurve->getPoint(0, Mbar_start, theta_start);
  auto Mbar_end = 0., theta_end = 0.;
  pMbarThetaCurve->getPoint(1., Mbar_end, theta_end);

  // 2.1 init
  auto np = NUMPTS;
  auto du = (u2 - u1) / (np - 1.);
  auto slop = 0.;
  QVector<Double2> mBeta; mBeta.resize(np);
  QVector<Double2> TP; TP.resize(np);
  QVector<double> Phi;

  // 2.2 loop
  for (auto i = 0; i < np; i++)
  {
    // 2.2.1 get ui/m
    auto ui = u1 + du * i;
    auto m = pZRCurve->calculate2DLength(u1, ui);

    // 2.2.2 getmBeta
    // a.calculate Mbar
    auto Mbar = cal_Mbar_conformalMapping(pMeanContour, pZRCurve, ule_mean, ui);

    // b.get betabr
    if (Mbar < Mbar_start)
      pMbarThetaCurve->getSlope(0., slop);
    else if (Mbar > Mbar_end)
      pMbarThetaCurve->getSlope(1., slop);
    else
    {
      auto found = false;
      auto u = pMbarThetaCurve->getUfromX(Mbar, found);
      if (found)
        pMbarThetaCurve->getSlope(u, slop);
      else
        eprintf("can't find Mbar parameters");
    }
```

```
    // c.Phi
    pZRCurve->getTangentialDirection(ui, TP[i]);
    Phi.push_back(TP[i].angle());

    // d.mBeta
    auto betabr = atan(slop);
    auto betabm = getbetam(betabr, Phi[i]);
    mBeta[i] = Double2(m, betabm);
  }

  // 3.get pmBetaCurve
  QVector<double> ms, Betab;
  for (auto i = 0; i < np; i++)
  {
    ms.push_back(mBeta[i][0]);
    Betab.push_back(mBeta[i][1]);
  }
  pmBetaCurve->fitBezier(&ms[0], &Betab[0], np, 8, 1, 3);
}

double T1dImpeller::cal_Mbar_conformalMapping(TNurbsCurve* pZRCurve1, TNurbsCurve* pZRCurve2, double u1, double u2)
{
  auto Z1 = 0., R1 = 0., Z2 = 0., R2 = 0.;
  pZRCurve1->getPoint(u1, Z1, R1);
  pZRCurve2->getPoint(u2, Z2, R2);
  return log(R2 / R1);
}
```