```
/*
 *
 *  This file is part of the Virtual Leaf.
 *
 *  The Virtual Leaf 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 Virtual Leaf is distributed in the hope that it will
be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied
warranty of
 *  MERCHANTABILITY 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 Virtual Leaf.  If not, see
<http://www.gnu.org/licenses/>.
 *
 *  Copyright 2010 Roeland Merks.
 *
 */

#include <QObject>
#include <QtGui>

#include "simplugin.h"

#include "parameter.h"
#include "wallbase.h"
#include "cellbase.h"
#include "tutorial0.h"
#include "far_mem_5.h"

static const std::string _module_id("$Id: tutorial0.cpp,v
6bcb69712a0e 2010/11/24 17:02:13 roeland $");

QString Tutorial0::ModelID(void) {
  // specify the name of your model here
  return QString( "Test_Model" );
}

// return the number of chemicals your model uses
int Tutorial0::NChem(void) { return 2; }

// To be executed after cell division
void Tutorial0::OnDivide(ParentInfo *parent_info, CellBase
*daughter1, CellBase *daughter2) {
  // rules to be executed after cell division go here
 // (e.g., cell differentiation rules)
// Auxin distributes between parent and daughter according to
```

```
area
  double area1 = daughter1->Area(), area2 = daughter2->Area();
  double tot_area = area1 + area2;

  daughter1->SetChemical(0,daughter1->
Chemical(0)*(area1/tot_area));
  daughter2->SetChemical(0,daughter2->
Chemical(0)*(area2/tot_area));

  // After divisions, parent and daughter cells get a standard
stock of PINs.
  daughter1->SetChemical(1, par->initval[1]);
  daughter2->SetChemical(1, par->initval[1]);


  // Reset transporter values of parent and daughter
  QList<WallBase *> walls;
  foreach(WallBase *w, walls) {
    w->setTransporter(daughter1, 0., 0.);
  }
}

void Tutorial0::SetCellColor(CellBase *c, QColor *color) {
    // add cell coloring rules here
   // Red: PIN1
  // Green: Auxin
  color->setRgb(c->Chemical(0)/(1+c->Chemical(0)) * 255.,(c->
Chemical(1)/(1+c->Chemical(1)) * 255.), 0);
}


void Tutorial0::CellHouseKeeping(CellBase *c) {
  // add cell behavioral rules here
if (c->Boundary()==CellBase::None) {
    if (c->Area() > par->rel_cell_div_threshold * c->
BaseArea() ) {
      c->SetChemical(0,0);
      c->Divide();
    }
    // expand according to substrate concentration

c->EnlargeTargetArea(par->auxin_dependent_growth?(c->
Chemical(0)/(1.+c->Chemical(0)))*(c->Chemical(1)/(1.+c->
Chemical(1)))*par->cell_expansion_rate:par->
cell_expansion_rate);
  }
}

void Tutorial0::CelltoCellTransport(Wall *w, double *dchem_c1,
double *dchem_c2) {
  // add biochemical transport rules here
// leaf edge is const source of auxin
  // (Neumann boundary condition: we specify the influx)
  if (w->C2()->BoundaryPolP()) {
    if (w->AuxinSource()) {
      double aux_flux = (par->i5 - w->C2()->Chemical(0))*par->
leaf_tip_source * w->Length();
```

```
      double oxy_flux = (par->i4 - w->C2()->Chemical(1))*par->
leaf_tip_source *w->Length();
      dchem_c1[0]+= aux_flux;
      dchem_c1[1]+= oxy_flux;
return;
    } else {
      return;
    }
  }

  if (w->C1()->BoundaryPolP()) {

    if (w->AuxinSource()) {
      double aux_flux = (par->i5 - w->C1()->Chemical(0))*par->
leaf_tip_source * w->Length();
      double oxy_flux = (par->i4 - w->C1()->Chemical(1))*par->
leaf_tip_source * w->Length();
      dchem_c2[0] += aux_flux;
      dchem_c2[0] += oxy_flux;
      return;
    } else {

      if (w->AuxinSink()) {

      // efflux into Shoot Apical meristem
      // we assume all PINs are directed towards shoot apical
meristem
     dchem_c2[0] -= par->sam_efflux * w->C2()->Chemical(0) /
(par->ka + w->C2()->Chemical(0));

      return;
       } else
      return;
    }
  }


  // Passive fluxes (Fick's law)
  // only auxin flux now
  // flux depends on edge length and concentration difference

  double phi = w->Length() * ( par->D[0] ) * ( w->C2()->
Chemical(0) - w->C1()->Chemical(0) );
  double phi1= w->Length() * ( par->D[1] ) * ( w->C2()->
Chemical(1) - w->C1()->Chemical(1) );

  dchem_c1[0] += phi ;
  dchem_c2[0] -= phi;
  dchem_c1[1] +=phi1;
  dchem_c2[1] -=phi1;

}
void Tutorial0::WallDynamics(Wall *w, double *dw1, double
*dw2) {
  // add biochemical networks for reactions occuring at walls
here
```

```
}

void Tutorial0::CellDynamics(CellBase *c, double *dchem) {
  // add biochemical networks for intracellular reactions here
 // source of auxin
    double rate = (c->Chemical(0)/(1. + c->Chemical(0))*(c->
Chemical(1)/(1. + c->Chemical(1)))*par->cell_expansion_rate);
    dchem[0] = -par->k1*rate - c->Chemical(0)*par->
aux_breakdown;
    dchem[1] = -par->k2*rate -c->Chemical(1)*par->
aux_breakdown;

  }



Q_EXPORT_PLUGIN2(tutorial0, Tutorial0)
```