```
#include "stdafx.h"
#include "msolidcreator.h"
#include "mcommandid.h"
#include "../maris/command.h"
#include "../maris/uimsg.h"
#include "../maris/mdoc.h"
#include "../MARIS/msolid.h"
#include "../muires/marismsg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

namespace maris
{

//////////////////////////////////////////////////////////////////////
// MSolidCreator factory object

static command::factory<MSolidCreator>                aCommandFactory;


//////////////////////////////////////////////////////////////////////
// MSolidCreator

MSolidCreator::MSolidCreator(MGeometryView* pView)
                        :                       MGeometryCreateCommand(pView),
                                                _pCurrentSolid(NULL),
                                                _pointCount(0),
                                                _RedoPointCountForIncompletedSolid(0),
                                                _RedoSolidsList(NULL)
{
                        _RedoPointsForIncompletedSolid.p1.set(0,0);
                        _RedoPointsForIncompletedSolid.p2.set(0,0);
                        _RedoPointsForIncompletedSolid.p3.set(0,0);
                        _RedoPointsForIncompletedSolid.p4.set(0,0);
}


MSolidCreator::~MSolidCreator()
{
                        if ( _pCurrentSolid )
                        {
                                                //delete _pCurrentSolid;
                                                _pCurrentSolid = NULL;
                        }
}

muint MSolidCreator::getId(void) const
{
                        return MCID_CREATE_SOLID;
}


/**
Method is not used in this class; returns a null pointer.
```

```
*/
MGeometryPointer MSolidCreator::getGeometry(void)
{
    return MGeometryPointer(NULL);
}

/**
Ends the commmand. Result is given in the parameter.
*/
void MSolidCreator::endCommand(MCEndResult Result)
{
    disconnect();

                        if (_pCurrentSolid)
                        {
        removeDrawables(_pCurrentSolid);
        update(VIEW_UPDATE_DRAWABLES);

                                            delete _pCurrentSolid;
                                            _pCurrentSolid = NULL;
                        }

    if ( Result == ABORT )
    {
        // Delete all created solids. This leaves them to the document, though...
        undo();

        // Flush pointers; no redo.
        _Solids.flush();
                                            _RedoSolidsList.clear();
    }

    // No view update.
                        MGeometryCreateCommand::end(Result, Result == SUCCESS, 0);
}


/**
Start command.Return SUCCESS if start is successfull. If False is returned, caller
must take care of destoying this object.
*/
MCEndResult MSolidCreator::start(void)
{
                        // Do base class start first
                        if ( MGeometryCreateCommand::start() == SUCCESS )
                        {
                                            // add command to command stack.
                                            MUICommand::stackCommand();

                                            // Connect to mouse, keyboard and input
                                            connect();

                                            // Show name of the command
        showName();

                                            // Start waiting point events.
        displayPrompt();
```

```
                // setup coordinate input state; we allways a point from user.
                setCoordinateInput();

  createObject();

                // set state to ACTIVE
                setState(ACTIVE);
                _SolidReady = false;

                return SUCCESS;
        }

        debugmsg("- Warning! MSolidCreator::start failed\n");

        return FAILURE;
}


/**
MInput point event handler.
*/
void MSolidCreator::point(void* pSrc, const Point3d& point3d)
{
   addPoint(Point2d(point3d));
}



mbool MSolidCreator::isLastUndoable() const
{
        if (_pointCount >= 1 || _Solids.getCount() > 0)
                return true;
        else
                return false;

}

mbool MSolidCreator::isLastRedoable() const
{
        if (_RedoSolidsList.size() > 0 || _RedoPointCountForIncompletedSolid > 0)
                return true;
        else
                return false;
}




mbool MSolidCreator::undoLast(void)
{
        if( _pointCount > 0 )
        {
                if (_pointCount == 2 && _Solids.getCount() > 0 ) // _pCurrentSolid kesken:
                {                                                                               // poistetaan viimeisin valmis solidi
(ja tehdään siitä rakenteilla oleva _pCurrentSolid)
                        Point2d points[4];
                        //Point2dArray pointsTest[4];
```

```
                        // poistetaan piirto tekeillä oleva solid (keskeneräinen)
                        removeDrawables(_pCurrentSolid);

                        // Poistetaan viimeisin valmis Solid sekä _Solids-taulukosta että dokumentista
                        mint k = _Solids.getCount();
                        MSolid* pRemovableSolid = _Solids.get(k-1);
                        // poistetaan viimeisimmän valmiin solidin piirto
                        removeDrawables(pRemovableSolid);
                        MDoc* pDoc = getDocument();
                        pDoc->erase(MGeometryPointer(pRemovableSolid)); // poistaa Solid:n dokumentista
                        _Solids.remove(k-1);
// ... ja valmiiden solidien listasta


                        // talletetaan viimeisin valmis solidi palauttamista varten RedoSolids-listaan
                        //pRemovableSolid->getValidPoints(*pointsTest);
                        mint r = pRemovableSolid->getPointCount();
                        SolidData sd;
                        sd.p1 = pRemovableSolid->getPoint(0);
                        sd.p2 = pRemovableSolid->getPoint(1);
                        sd.p3 = pRemovableSolid->getPoint(2);
                        if (r == 4)
                        {
                                        sd.p4 = pRemovableSolid->getPoint(3);
                                        sd.threePointSolid = false;
                        }
                        else
                        {
                                        // kolmen pisteen solid
                                        sd.p4 = Point2d(0,0);
                                        sd.threePointSolid = true;
                        }
                        _RedoSolidsList.push_back(sd);


                        // Poistetun Solidin koordinaatit asetetaan tekeillä olevan _pCurrentSolid:n arvoiksi

                        pRemovableSolid->getPoints(points);

                        _pCurrentSolid->setPoint(0, points[0]);
                        _pCurrentSolid->setPoint(1, points[1]);
                        _pCurrentSolid->setPoint(2, points[2]);
                        if (r == 4)
                        {
                                        _pCurrentSolid->setPoint(3, points[3]);
                                        _pointCount = 3;
                                        _pCurrentSolid->setPointCount(4);
                        }
                        else
                        {
                                        // kolmen pisteen solid
                                        _pCurrentSolid->setPoint(3, Point2d(0,0));
                                        _pointCount = 3;
                                        _pCurrentSolid->setPointCount(3);
                        }

                        addDrawables(_pCurrentSolid);
```

```
								showPrompt( PROMPT_GIVE_NEXT_POINT, _pointCount+1 );

								update(VIEW_UPDATE_DRAWABLES);
#ifdef _DEBUG
								debugPrintUndoRedoPoints();
#endif
								return true;
						}
						else if (_pointCount == 1) // Yksinkertaisin tilanne: yksi piste (=aloituspiste) annettu (ja hiiren liike määrittää toista pistettä)
																		// Undo tallettaa
aloituspisteen ja tallettaa sen mahdollista Redo-toimintoa varten
						{
								Point2d points[4];
								removeDrawables(_pCurrentSolid);
								_pCurrentSolid->getPoints(points);
								debugmsg2("Keskeneräinen Solid: (%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)\n",
										points[0].x, points[0].y,
										points[1].x, points[1].y,
										points[2].x, points[2].y,
										points[3].x, points[3].y );
								_pointCount--;
								_RedoPointCountForIncompletedSolid++;
								_RedoPointsForIncompletedSolid.p1.x = points[0].x;
								_RedoPointsForIncompletedSolid.p1.y = points[0].y;

								_pCurrentSolid->setPointCount(_pointCount);
								points[1].x = points[0].x;
								points[1].y = points[0].y;
								_pCurrentSolid->setPoint(1, points[1]);

								showPrompt( PROMPT_GIVE_NEXT_POINT, _pointCount );

								update(VIEW_UPDATE_DRAWABLES);
								//addDrawables(_pCurrentSolid);
								displayPrompt();

								return true;
						}
						else if (_pointCount == 2) // Kaksi pistettä annettu ja jälkimmäinen perutaan
						{
								Point2d points[4];
								removeDrawables(_pCurrentSolid);
								_pCurrentSolid->getPoints(points);
								debugmsg2("Keskeneräinen Solid: (%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)\n",
										points[0].x, points[0].y,
										points[1].x, points[1].y,
										points[2].x, points[2].y,
										points[3].x, points[3].y );
								_pointCount--;
								_RedoPointCountForIncompletedSolid++;
								_RedoPointsForIncompletedSolid.p2.x = points[1].x;
								_RedoPointsForIncompletedSolid.p2.y = points[1].y;

								_pCurrentSolid->setPointCount(_pointCount);
								points[1].x = points[0].x;
								points[1].y = points[0].y;
```

```
                                        _pCurrentSolid->setPoint(1, points[1]);

                                        showPrompt( PROMPT_GIVE_NEXT_POINT, _pointCount );

                                        update(VIEW_UPDATE_DRAWABLES);
                                        //addDrawables(_pCurrentSolid);
                                        displayPrompt();

                                        return true;
                                }
                                else if (_pointCount == 3) // 3 pistettä annettu ja viimeisin perutaan
                                {
                                        Point2d points[4];
                                        removeDrawables(_pCurrentSolid);
                                        _pCurrentSolid->getPoints(points);
                                        debugmsg2("Keskeneräinen Solid: (%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)\n",
                                                        points[0].x, points[0].y,
                                                        points[1].x, points[1].y,
                                                        points[2].x, points[2].y,
                                                        points[3].x, points[3].y );
                                        _pointCount--;
                                        _RedoPointCountForIncompletedSolid++;
                                        _RedoPointsForIncompletedSolid.p3.x = points[2].x;
                                        _RedoPointsForIncompletedSolid.p3.y = points[2].y;

                                        _pCurrentSolid->setPointCount(_pointCount);
                                        points[2].x = points[1].x;
                                        points[2].y = points[1].y;
                                        _pCurrentSolid->setPoint(2, points[2]);

                                        showPrompt( PROMPT_GIVE_NEXT_POINT, _pointCount );

                                        addDrawables(_pCurrentSolid);
                                        update(VIEW_UPDATE_DRAWABLES);
                                        displayPrompt();

                                        return true;
                                }

                        }
                        update(VIEW_UPDATE_DRAWABLES);
                        return false;
}

mbool MSolidCreator::redoLast(void)
{
                        if (_pointCount == 0) // alkupiste on annettu mutta peruttu Undo:lla
                                                                                                                // _RedoPointsForIncompletedSolid sisältää yhden validin palautettavan
pisteen (_RedoPointCountForIncompletedSolid = 1)
                        {
                                        Point2d points[4];
                                        removeDrawables(_pCurrentSolid);

                                        points[0].x = _RedoPointsForIncompletedSolid.p1.x;    points[0].y = _RedoPointsForIncompletedSolid.p1.y;
                                        points[1].x = 0;    points[1].y = 0;
                                        points[2].x = 0;    points[2].y = 0;
                                        points[3].x = 0;    points[3].y = 0;
```

```
                                        _pCurrentSolid->setPoints(points);
                                        debugmsg2("Keskeneräinen Solid: (%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)\n",
                                                  points[0].x, points[0].y,
                                                  points[1].x, points[1].y,
                                                  points[2].x, points[2].y,
                                                  points[3].x, points[3].y );
                                        _pointCount++;
                                        _RedoPointCountForIncompletedSolid--;
                                        _pCurrentSolid->setPointCount(_pointCount);
                                        points[1].x = points[0].x;
                                        points[1].y = points[0].y;
                                        _pCurrentSolid->setPoint(1, points[1]);

                                        showPrompt( PROMPT_GIVE_NEXT_POINT, _pointCount );

                                        addDrawables(_pCurrentSolid);
                                        update(VIEW_UPDATE_DRAWABLES);
                                        displayPrompt();

                                        return true;
                    }
                    else if (_pointCount == 1) // kaksi pistettä, alkupiste ja toinen piste, annettu - jälkimmäinen piste peruttu Undolla
                    {                                                                                                   // _RedoPointsForIncompletedSolid sisältää yhden
validin palautettavan pisteen (_RedoPointCountForIncompletedSolid = 1)
                                                                                                                        // alkupiste säilyy ennallaan mutta toinen piste
palautetaan tallennetuista (_RedoPointsForIncompletedSolid)
                                        Point2d points[4];
                                        removeDrawables(_pCurrentSolid);

                                        _pCurrentSolid->getPoints(points);
                                        //points[0].x = _RedoPointsForIncompletedSolid.p1.x; points[0].y = _RedoPointsForIncompletedSolid.p1.y;
                                        points[1].x = _RedoPointsForIncompletedSolid.p2.x;                          points[1].y = _RedoPointsForIncompletedSolid.p2.y;
                                        points[2].x = 0;    points[2].y = 0;
                                        points[3].x = 0;    points[3].y = 0;

                                        _pCurrentSolid->setPoints(points);
                                        debugmsg2("Keskeneräinen Solid: (%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)\n",
                                                  points[0].x, points[0].y,
                                                  points[1].x, points[1].y,
                                                  points[2].x, points[2].y,
                                                  points[3].x, points[3].y );
                                        _pointCount++;
                                        _RedoPointCountForIncompletedSolid--;
                                        _pCurrentSolid->setPointCount(_pointCount);
                                        points[2].x = points[1].x;
                                        points[2].y = points[1].y;
                                        _pCurrentSolid->setPoint(2, points[2]);

                                        showPrompt( PROMPT_GIVE_NEXT_POINT, _pointCount );

                                        addDrawables(_pCurrentSolid);
                                        update(VIEW_UPDATE_DRAWABLES);
                                        displayPrompt();

                                        return true;
                    }
                    else if (_pointCount == 2) // kolme pistettä annettu ja viimeinen piste on peruttu Undo:lla -> kolmas piste palautetaan
                    {
```

```
                Point2d points[4];
                removeDrawables(_pCurrentSolid);

                _pCurrentSolid->getPoints(points);
                //points[0].x = _RedoPointsForIncompletedSolid.p1.x; points[0].y = _RedoPointsForIncompletedSolid.p1.y;
                //points[1].x = _RedoPointsForIncompletedSolid.p2.x;                    points[1].y = _RedoPointsForIncompletedSolid.p2.y;
                points[2].x = _RedoPointsForIncompletedSolid.p3.x;    points[2].y = _RedoPointsForIncompletedSolid.p3.y;
                points[3].x = 0;    points[3].y = 0;

                _pCurrentSolid->setPoints(points);
                debugmsg2("Keskeneräinen Solid: (%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)\n",
                                points[0].x, points[0].y,
                                points[1].x, points[1].y,
                                points[2].x, points[2].y,
                                points[3].x, points[3].y );
                _pointCount++;
                _RedoPointCountForIncompletedSolid--;
                _pCurrentSolid->setPointCount(_pointCount);
                points[3].x = points[2].x;
                points[3].y = points[2].y;
                _pCurrentSolid->setPoint(2, points[2]);

                showPrompt( PROMPT_GIVE_NEXT_POINT, _pointCount );

                addDrawables(_pCurrentSolid);
                update(VIEW_UPDATE_DRAWABLES);
                displayPrompt();

                return true;
        }
        else if (_pointCount == 3 && _RedoSolidsList.size() > 0) // Ainakin yksi valmis Solid:

                                                                                // seuraavan solidin piirto (CurrentSolid) on kesken (=kysyy kolmatta
pistettä)

                                                                                // Undo tehty (= Valmis solid poistettun ja purettu keskeneräiseksi
(CurrentSolid))

                                                                                // => Redo => RedoSolidListalta haetaan aiemmin Undolla talletettu
valmis solid

                                                                                // ja lisätään se valmiiden solidien listaan ja dokumenttiin
        {
                MDoc* pDoc = getDocument();
                Point2d points[4];

                // 1. poistetaan väliaikainen drawables-piirto-objekti
                removeDrawables(_pCurrentSolid);


                // 2) + 4) Palautettava Solid haetaan ja poistetaan RedoSolid-listalta
                SolidData sd = _RedoSolidsList[_RedoSolidsList.size()-1];
                _RedoSolidsList.pop_back();

                points[0].x = sd.p1.x;
                points[0].y = sd.p1.y;
                points[1].x = sd.p2.x;
                points[1].y = sd.p2.y;
```

```
                                        points[2].x = sd.p3.x;
                                        points[2].y = sd.p3.y;
                                        points[3].x = sd.p4.x; // = 0 kolmen pisteen solidilla
                                        points[3].y = sd.p4.y;
                                        if (sd.threePointSolid == false)
                                        {
                                                        // CurrentSolid asetetaan ja käytetään väliaikaisesti
                                                        _pCurrentSolid->setPointCount(4);
                                                        _pCurrentSolid->setPoints(points);
                                                        _pointCount = 4;
                                        }
                                        else // kolmen pisteen solid
                                        {
                                                        _pCurrentSolid->setPoints(points);
                                                        _pCurrentSolid->setPointCount(3);
                                                        _pointCount = 3;
                                        }

                                        // 3) lisätään palautettu vanha solid dokumentille ja Solid-listalle
                                        //MDoc* pDoc = getDocument(); // ei tarvita koska addCurrentSolid hoitaa homman
                                        //pDoc->add(_pCurrentSolid);

                                        addCurrentSolid(true);


                                        addDrawables(_pCurrentSolid);


#ifdef _DEBUG
                                        debugPrintUndoRedoPoints();
#endif


                                        debugmsg2("Keskeneräinen (%d-pistettä) Solid: (%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)\n",
                                                        _pCurrentSolid->getPointCount(),
                                                        points[0].x, points[0].y,
                                                        points[1].x, points[1].y,
                                                        points[2].x, points[2].y,
                                                        points[3].x, points[3].y );


                                        showPrompt( PROMPT_GIVE_NEXT_POINT, _pointCount );

                                        addDrawables(_pCurrentSolid);
                                        update(VIEW_UPDATE_DRAWABLES);
                                        displayPrompt();

                                        return true;
                        }

                        return false;
}



void MSolidCreator::move(void* pSrc, const Point3d& wcsPosition, muint keys)
{
```

```
    setCurrentPoint(Point2d(wcsPosition));
}


void MSolidCreator::cancelMouse(void* pSrc, const Point3d& wcsPosition, muint keys)
{
                        // End command with mouse cancel
    //

    // Jos tämä on ensimmäinen solid, lisätään se. Jos taas ei ole, solidia ei pidä
    // lisätä, koska ensimmäisen jälkeen uudessa on aina kaksi pistettä valmiina, ja
    // jos käyttäjä haluaa lisätä sen, hänen pitää hyväksyä se enterillä.
    if ( _Solids.getCount() == 0 )
        addCurrentSolid(False);

    endCommand(SUCCESS);
}


/**
Selection (Enter): Accept current solid.
*/
imKeyboard::Result MSolidCreator::selectKb(void* pSrc)
{
    if ( _pointCount > 1 )
    {
        addCurrentSolid(True);

        displayPrompt();
    }

    return imKeyboard::Used;
}


void MSolidCreator::abort(void)
{
    endCommand(ABORT);
}


void MSolidCreator::activate(MCEndResult previous_command_result)
{
    displayPrompt();

                        MGeometryCreateCommand::activate(previous_command_result);

                        // setup coordinate input state; we allways a point from user.
                        setCoordinateInput();
}


/**
Returns array of the created solids.

This method is intented to be used only when created object is
not stored to document.
*/
```

```
const MArray<MSolid*> MSolidCreator::getSolids(void) const
{
                        return _Solids;
}

void MSolidCreator::createObject(void)
{
   if ( !_pCurrentSolid )
   {
                           _pCurrentSolid = new MSolid;

      // set default values
      setCurrentValues(_pCurrentSolid);
   }
}


/**
Adds a new point to the solid.
*/
void MSolidCreator::addPoint(const Point2d& Point)
{
   massert(_pCurrentSolid);
   debugmsg2(L"addPoint: %.1f,%.1f (Yht %d pistettä)\n", Point.x, Point.y, _pointCount);

   if ( _pCurrentSolid->setPoint(_pointCount, Point) )
   {
      _pointCount++;

      // Jos valmis, aloitetaan uusi
                                       if ( _pointCount == 4 )
                                       {
                                                               _SolidReady = true;
         addCurrentSolid(True);
                                       }
      else
      {
         _SolidReady = false;
                                                               // add next point for visual clue
         _pCurrentSolid->setPoint(_pointCount, Point);

         // Solid voidaan piirtää vasta kun on annettu vähintään kaksi pistettä.
         if ( _pointCount == 2 )
         {
                                             // Connect to view
                                             addDrawables(_pCurrentSolid);
                                             connectViews();

         }
      }
                                       // clear Undo/Redo-history
                                       _RedoSolidsList.clear();
                                       _RedoPointsForIncompletedSolid.p1.set(0,0);
                                       _RedoPointsForIncompletedSolid.p2.set(0,0);
                                       _RedoPointsForIncompletedSolid.p3.set(0,0);
                                       _RedoPointsForIncompletedSolid.p4.set(0,0);
                                       _RedoPointCountForIncompletedSolid = 0;
   }
   else
```

```
        UImsg(IDS_ERROR_POINT_NOT_ACCEPTED, M_OUTPUT);

                        debugmsg2(L"addPoint: %.1f,%.1f (Yht %d pistettä)\n", Point.x, Point.y, _pointCount);

                        displayPrompt();
}


/**
Sets a "current" point, eq. point form mouse move.
*/
void MSolidCreator::setCurrentPoint(const Point2d& Point)
{
                        if ( _pCurrentSolid && _pointCount > 1 )
                        {
                                        // Update polyline; set current point to new point.
                                        _pCurrentSolid->setPoint(_pointCount, Point);

                                        // Do a fast update.
                                        update(VIEW_UPDATE_DRAWABLES);
                        }
}


/**
Adds current solid to the document. Prepares a new solid for editing,
if startNew == True.
*/
void MSolidCreator::addCurrentSolid(mbool startNew)
{
    massert(_pCurrentSolid);
    if ( _pCurrentSolid && _pointCount > 1 )
    {
        if ( _pointCount < 4 )
            _pCurrentSolid->setPointCount(_pointCount);

                        removeDrawables(_pCurrentSolid);

        if ( _pCurrentSolid->isValid() )
        {
            MDoc* pDoc = getDocument();
                                if ( pDoc )
                                {
                // Viimeiset kaksi pistettä otetaan seuraavan solidin pisteiksi.
                Point2d last[2];
                last[0] = _pCurrentSolid->getPoint(_pointCount-2);
                last[1] = _pCurrentSolid->getPoint(_pointCount-1);

                // Use may have changed linetype, color etc. during creation.
                setCurrentValues(_pCurrentSolid);

                _Solids.add(_pCurrentSolid);
                pDoc->add(_pCurrentSolid);

                _pCurrentSolid = NULL;
                _pointCount = 0;

                if ( startNew )
```

```
            {
               createObject();

               _pCurrentSolid->setPoint(0, last[0]);
               _pCurrentSolid->setPoint(1, last[1]);
               _pointCount = 2;

               addDrawables(_pCurrentSolid);
            }

            update(VIEW_UPDATE_CREATE);
         }
      }
      else
      {
         UImsg(IDS_ERROR_INVALID_POINT_OBJECT, M_OUTPUT);

         // Vanhaa objektia ei voi oikein uudelleen käyttää (do:t)
         delete _pCurrentSolid;
         _pCurrentSolid = NULL;
         _pointCount = 0;

         if ( startNew )
            createObject();
      }
   }
}

void MSolidCreator::displayPrompt(void)
{
   showPrompt( (_pointCount > 1 ) ? PROMPT_GIVE_NEXT_POINT_OR_ACCEPT : PROMPT_GIVE_NEXT_POINT,
            _pointCount+1 );
}


void MSolidCreator::cancel( void* pSrc, mbool TotalCancel )
{
   abort();
}

mbool MSolidCreator::isUndoable(void) const
{
                     return !_Solids.isEmpty();
}


mbool MSolidCreator::undo(void)
{
                     // Use document's method for removal, if the object
                     // is in the document. If not, there's nothing to undo.
                     MDoc* pDoc = getDocument();
                     if ( pDoc )
                     {
      // Delete all created solids
      for ( MArray<MSolid*>::stl_iterator i = _Solids.begin(); i != _Solids.end(); ++i )
      {
         MSolid* pSolid = (*i);
         if ( pSolid )
```

```
        {
            // remove selection
            if ( pSolid->isSelected(NULL) )
                pDoc->deselectObject(MGeometryPointer(pSolid), false);

                                                    if ( !pDoc->erase(MGeometryPointer(pSolid)) )
                                                                    return false;
        }
    }

                                            // removal ok, update view(s)
                                            update(VIEW_UPDATE_DELETE);
                    }

                    return true;
}


mbool MSolidCreator::redo(void)
{
    MDoc* pDoc = getDocument();
    if ( !pDoc || _Solids.isEmpty() ) return False;

    // Restore all created solids
    for ( MArray<MSolid*>::stl_iterator i = _Solids.begin(); i != _Solids.end(); ++i )
    {
        MSolid* pSolid = (*i);
        if ( pSolid )
        {
                            // Because object is marked as deleted, no ownership
                            // can be taken. So, just write directly to object.
            // Use document's 'restore' method.

                                                                    pDoc->restore(MGeometryPointer(pSolid));
        }
    }

    update(VIEW_UPDATE_UNDELETE);

                    return True;
}

#ifdef _DEBUG
void MSolidCreator::debugPrintUndoRedoPoints(void)
{
                    debugmsg2("-> ------------------------------------------------\n");
                    if (_pCurrentSolid->isValid())
                    {
                                            debugmsg2("-> debugPrint(): %d pistettä ",  _pCurrentSolid->getPointCount());
                    }
                    debugmsg2("(_pointCount = %d)\n", _pointCount);

                    MDoc* pDoc = getDocument();

                    /*MArray<MSolid*>::stl_iterator i = _Solids.begin();
                    while ( i != _Solids.end())
                    {
                    }*/
```

```
Point2d points[4];
_pCurrentSolid->getPoints(points);
debugmsg2("Keskeneräinen Solid: (%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)(%4.1f, %4.1f)\n",
                                                                                                                    points[0].x, points[0].y,
                                                                                                                    points[1].x, points[1].y,
                                                                                                                    points[2].x, points[2].y,
                                                                                                                    points[3].x, points[3].y );

// Tulostaa -Solids -taulukon sisällön
debugmsg2("-> debugPrint(): %d valmista solid:ia (_Solids.getCount()\n", _Solids.getCount());
mint t = 0;
for ( MArray<MSolid*>::stl_iterator i = _Solids.begin(); i != _Solids.end(); i++ )
{
                    Point2d points[4];


                    //_Solids.getCount();
                    //MSolid* pCurSolid = _Solids.get(t);
                    MSolid* pCurSolid = *i;
                    mint pnt = pCurSolid->getPointCount();
                    pCurSolid->getPoints(points);

                    debugmsg2("-> debugPrint(): _Solids%d (%4.1f, %4.1f), (%4.1f, %4.1f), (%4.1f, %4.1f), (%4.1f, %4.1f)\n", t+1,


                                                                points[0].x, points[0].y,


                                                                points[1].x, points[1].y,


                                                                points[2].x, points[2].y,


                                                                points[3].x, points[3].y );
                    debugmsg2("-> debugPrint(): Solidissa pisteitä %d\n", pCurSolid->getPointCount());
                    massert (pnt == 3 || pnt == 4);
                    t++;
}
// Tulostaa -RedoSolidsList -sisällön
debugmsg2("->                 ----------_RedoSolidsList ----------\n");
debugmsg2("->                 debugPrint(): %d RedoSolid:ia (_RedoSolidsList.size()\n", _RedoSolidsList.size());

t = 1;
for ( std::vector<SolidData>::iterator i = _RedoSolidsList.begin(); i != _RedoSolidsList.end(); i++ )
{
                    Point2d points[4];

                    SolidData pCurSolid = *i;
                    points[0] = pCurSolid.p1;
                    points[1] = pCurSolid.p2;
                    points[2] = pCurSolid.p3;
                    points[3] = pCurSolid.p4;
```

```
                                debugmsg2("->                debugPrint(): _RedoSolidsList[%d] (%4.1f, %4.1f), (%4.1f, %4.1f), (%4.1f, %4.1f), (%4.1f, %4.1f)\n", t,

                                                                        points[0].x, points[0].y,

                                                                        points[1].x, points[1].y,

                                                                        points[2].x, points[2].y,

                                                                        points[3].x, points[3].y );
                                t++;
                        }
                        debugmsg2("->                debugPrint():  ----- keskeneräinen Redo-solid (Redo=%d) --------\n", _RedoPointCountForIncompletedSolid);

                        debugmsg2("(%4.1f, %4.1f), (%4.1f, %4.1f), (%4.1f, %4.1f), (%4.1f, %4.1f)\n",
                                _RedoPointsForIncompletedSolid.p1.x, _RedoPointsForIncompletedSolid.p1.y,
                                _RedoPointsForIncompletedSolid.p2.x, _RedoPointsForIncompletedSolid.p2.y,
                                _RedoPointsForIncompletedSolid.p3.x, _RedoPointsForIncompletedSolid.p3.y,
                                _RedoPointsForIncompletedSolid.p4.x, _RedoPointsForIncompletedSolid.p4.y
                                );


}

#endif

} // namespace maris
```