```cpp
#include "StdAfx.h"
#include "SMPBase.h"

#include <PArray.h>
#include <SafeArrayUtils.h>

extern int LibMessage(LPCTSTR str, int flags = MB_OK);
extern IDocumentFramePtr GetActiveFrame(IKompasDocumentPtr& doc);

void SMPBase::init(IApplicationPtr application, HMODULE hmodule) {
    this->application = application;
    this->hmodule = hmodule;
    this->initDocumentParameters();
}

void SMPBase::initDocumentParameters() {
    this->active_document = this->application->ActiveDocument;
    this->k_document_3d = IKompasDocument3DPtr(this->active_document);
    this->doc3D = ksGetActive3dDocument();

    IPartPtr p = NULL;
    if (this->doc3D) {
        p = this->doc3D->GetPart(Part_Type::pTop_Part);
    }
    this->part = IPartPtr(p);
    this->part7 = this->k_document_3d->TopPart;
    this->model_container = IModelContainerPtr(this->part7);
}

double SMPBase::getOuterRadius() {
    return this->outerRadius;
}

double SMPBase::getInnerRadius() {
    return this->innerRadius;
}

double SMPBase::getSide() {
    return this->side;
}

double SMPBase::getHeight() {
    return (this->height + this->y_delta);
}

unsigned short SMPBase::getNGonSideCount() {
    unsigned short count = 0;
    if (this->isTriangle())
        count = 3;
    else if (this->isSquare())
        count = 4;
    else if (this->isPentagon())
        count = 5;
    else if (this->isHexagon())
        count = 6;
    return count;
}
```

```cpp

double SMPBase::getRhombusAngle() {
    double angle = 0;
    if (this->isRhombus35())
        angle = 35;
    else if (this->isRhombus55())
        angle = 55;
    else if (this->isRhombus80())
        angle = 80;
    return angle;
}

void SMPBase::getPoint(size_t index, double* x, double* y) {
    size_t size = this->points.size();
    if (size && (index >= 0) && (index < size)) {
        *x = get<0>(this->points[index]);
        *y = get<1>(this->points[index]);
    }
}

void SMPBase::getPointFirst(double* x, double* y) {
    this->getPoint(0, x, y);
}

void SMPBase::getPointLast(double* x, double* y) {
    size_t last = this->points.size() - 1;
    this->getPoint(last, x, y);
}

void SMPBase::addPoint(double x, double y) {
    this->points.push_back(make_tuple(x, y));
}

bool SMPBase::isByInnerRadius() {
    return (this->sizeType == SizeType::byInnerRadius);
}

bool SMPBase::isByOuterRadius() {
    return (this->sizeType == SizeType::byOuterRadius);
}

bool SMPBase::isBySide() {
    return (this->sizeType == SizeType::bySide);
}

bool SMPBase::isTriangle() {
    return (this->surfaceType == SurfaceType::triangle);
}

bool SMPBase::isSquare() {
    return (this->surfaceType == SurfaceType::square);
}

bool SMPBase::isPentagon() {
    return (this->surfaceType == SurfaceType::pentagon);
}

bool SMPBase::isHexagon() {
    return (this->surfaceType == SurfaceType::hexagon);
}

bool SMPBase::isRhombus35() {
    return (this->surfaceType == SurfaceType::rhombus35);
}

bool SMPBase::isRhombus55() {
    return (this->surfaceType == SurfaceType::rhombus55);
}

bool SMPBase::isRhombus80() {
    return (this->surfaceType == SurfaceType::rhombus80);
}

bool SMPBase::isTrigon() {
    return (this->surfaceType == SurfaceType::trigon);
}

bool SMPBase::isCircle() {
    return (this->surfaceType == SurfaceType::circle);
}

bool SMPBase::isNGon() {
    return (this->isTriangle() ||
        this->isSquare() ||
        this->isPentagon() ||
        this->isHexagon());
}

bool SMPBase::isRhombus() {
    return (this->isRhombus35() ||
        this->isRhombus55() ||
        this->isRhombus80());
}

bool SMPBase::checkHasHole() {
    return (this->hasHole && (this->holeRadius > 0));
}

IDrawingContainerPtr SMPBase::getDrawingContainer(IFragmentDocumentPtr sketch_document) {
    IKompasDocument2DPtr doc2d(sketch_document);
    IViewPtr view = doc2d->ViewsAndLayersManager->Views->ActiveView;
    IDrawingContainerPtr drawing_container(view);
    return drawing_container;
}

void SMPBase::addEmbodiment() {
    IEmbodimentsManagerPtr eMng(this->part7);
    eMng->TopEmbodiment->IsCurrent = true;
    long count = eMng->EmbodimentCount;
    long top_index = eMng->CurrentEmbodimentIndex;

    wchar_t temp_str[11];
    _ltow(count, temp_str, 10);
    BSTR embodiment_number = SysAllocString(temp_str);
    BSTR base_marking = _T("СМП");
    BSTR additional_number = _T("1");
    eMng->AddEmbodiment(
        top_index, false, base_marking,
        embodiment_number, additional_number
    );
    this->initDocumentParameters();
}

void SMPBase::drawBase() {
    this->base_plane = this->model_container->AddObject(ksObj3dTypeEnum::o3d_planeOffset);
    this->base_plane->Name = _T("Плоскость основания СМП");
    this->base_plane->BasePlane = this->part7->GetDefaultObject(ksObj3dTypeEnum::o3d_planeXOY);
    this->base_plane->Offset = this->coordinates.Z;
    this->base_plane->Update();

    this->base_sketch = this->model_container->Sketchs->Add();
    this->base_sketch->Name = _T("Основание СМП");
    this->base_sketch->Plane = this->base_plane;
    IFragmentDocumentPtr sketch_document = this->base_sketch->BeginEdit();
    IDrawingContainerPtr drawing_container = this->getDrawingContainer(sketch_document);
    if (this->isCircle()) {
        this->drawCircle(drawing_container);
    }
    else if (this->isNGon()) {
        this->drawNGon(drawing_container);
    }
    else if (this->isTrigon()) {
        this->drawTrigon(drawing_container);
    }
    else if (this->isRhombus()) {
        this->drawRhombus(drawing_container);
    }
    this->base_sketch->EndEdit();
}

void SMPBase::drawCircle(IDrawingContainerPtr drawing_container) {
    ICirclePtr figure = drawing_container->Circles->Add();
    figure->Style = ksCurveStyleEnum::ksCSNormal;
    figure->Xc = this->coordinates.X;
    figure->Yc = this->coordinates.Y;
    figure->Radius = this->size;
    figure->Update();

    this->addPoint(
        this->coordinates.X + this->size,
        this->coordinates.Y
    );
}

void SMPBase::drawNGon(IDrawingContainerPtr drawing_container) {
    unsigned short count = this->getNGonSideCount();
    double angle_vertex = 360.0 / count;
    this->base_sketch->Angle = 180.0;

    IRegularPolygonPtr figure = drawing_container->RegularPolygons->Add();
    figure->Count = count;
    figure->Style = ksCurveStyleEnum::ksCSNormal;
    figure->Xc = this->coordinates.X;
    figure->Yc = this->coordinates.Y;
    figure->Radius = this->innerRadius;
    figure->Describe = true;
    figure->Update();

    double angle_i;
    for (size_t i = 0; i < count; i++) {
        angle_i = angle_vertex * i + angle_vertex / 2;
        this->addPoint(
            this->outerRadius * cosd(angle_i),
            this->outerRadius * sind(angle_i)
        );
    }
}

void SMPBase::drawTrigon(IDrawingContainerPtr drawing_container) {
    double delta_max = this->innerRadius / sind(40);
    double delta_min = this->innerRadius / sind(80);

    this->addPoint(
        this->coordinates.X,
        this->coordinates.Y + delta_max
    );
    this->addPoint(
        this->coordinates.X + delta_min * sind(60),
        this->coordinates.Y + delta_min * sind(30)
    );
    this->addPoint(
        this->coordinates.X + delta_max * sind(60),
        this->coordinates.Y - delta_max * sind(30)
    );
    this->addPoint(
        this->coordinates.X,
        this->coordinates.Y - delta_min
    );
    this->addPoint(
        this->coordinates.X - delta_max * sind(60),
        this->coordinates.Y - delta_max * sind(30)
    );
    this->addPoint(
        this->coordinates.X - delta_min * sind(60),
        this->coordinates.Y + delta_min * sind(30)
    );

    IPolyLine2DPtr figure = drawing_container->PolyLines2D->Add();
    figure->Style = ksCurveStyleEnum::ksCSNormal;
    figure->Closed = true;
    double x, y;
    for (size_t i = 0; i < 6; i++) {
        this->getPoint(i, &x, &y);
        figure->AddPoint(i, x, y);
    }
    figure->Update();
}

void SMPBase::drawRhombus(IDrawingContainerPtr drawing_container) {
    double angle = this->getRhombusAngle() / 2;
    double delta_min = this->innerRadius / cosd(angle);
    double delta_max = this->innerRadius / sind(angle);

    this->addPoint(
        this->coordinates.X,
        this->coordinates.Y + delta_max
    );
    this->addPoint(
        this->coordinates.X + delta_min,
        this->coordinates.Y
    );
    this->addPoint(
        this->coordinates.X,
        this->coordinates.Y - delta_max
    );
    this->addPoint(
        this->coordinates.X - delta_min,
        this->coordinates.Y
    );

    IPolyLine2DPtr figure = drawing_container->PolyLines2D->Add();
    figure->Style = ksCurveStyleEnum::ksCSNormal;
    figure->Closed = true;
    double x, y;
    for (size_t i = 0; i < 4; i++) {
        this->getPoint(i, &x, &y);
        figure->AddPoint(i, x, y);
    }
    figure->Update();
}

void SMPBase::addExtrusion() {
    IExtrusionPtr extrusion = this->model_container->Extrusions->Add(ksObj3dTypeEnum::o3d_bossExtrusion);
    extrusion->Name = _T("Элемент выдавливания СМП");
    extrusion->Direction = ksDirectionTypeEnum::dtNormal;
    extrusion->SetSideParameters(
        true, ksEndTypeEnum::etBlind,
        this->getHeight(), 0, true, NULL
    );
    extrusion->PutDraftValue(true, this->angleAlpha);
    extrusion->Sketch = this->base_sketch;
    extrusion->Update();
}

void SMPBase::addRounding() {
    if (this->isCircle() || !this->roundingRadius) {
        return;
    }
    double z1 = this->coordinates.Z;
    double z2 = z1 + this->getHeight();
    IFilletPtr fillet = this->model_container->AddObject(Obj3dType::o3d_fillet);
    fillet->Name = _T("Скругление боковых рёбер пластины");
    fillet->Radius1 = this->roundingRadius;
    fillet->BuildingType = ksFilletBuildingTypeEnum::ksFilletCircleArc;
    PArray<IModelObjectPtr> fillet_objects;

    IFeature7Ptr feature(this->part7);
    _variant_t faces = feature->GetModelObjects(Obj3dType::o3d_face);
    LPDISPATCH* pFaces = NULL;
    long flCount, flBound, fuBound;
    setSafeArrayParameters(&faces, &pFaces, &flCount, &flBound, &fuBound);
    for (long idf = 0; idf < flCount; idf++) {
        IFacePtr face = pFaces[idf];
        if (face->BaseSurface3DType == ksMathSurface3DTypeEnum::ksPlane) {
            double s_x1, s_y1, s_z1, s_x2, s_y2, s_z2;
            face->GetMathSurface()->GetGabarit(&s_x1, &s_y1, &s_z1, &s_x2, &s_y2, &s_z2);
            if ((round(s_z1) == round(z1)) &&
                (round(s_z2) == round(z2))) {
                _variant_t edges = face->GetLimitingEdges();
                LPDISPATCH* pEdges = NULL;
                long elCount, elBound, euBound;
                setSafeArrayParameters(&edges, &pEdges, &elCount, &elBound, &euBound);
                for (long ide = 0; ide < elCount; ide++) {
                    IEdgePtr edge = pEdges[ide];
                    double c_x1, c_y1, c_z1, c_x2, c_y2, c_z2;
                    edge->GetMathCurve()->GetGabarit(&c_x1, &c_y1, &c_z1, &c_x2, &c_y2, &c_z2);
                    if (round(c_z1) != round(c_z2)) {
                        fillet_objects.Add(new IModelObjectPtr(edge));
                    }
                }
            }
        }
    }
    fillet->BaseObjects = CreateDispSafeArray(fillet_objects);
    fillet->Update();
}

void SMPBase::addHole() {
    if (!this->checkHasHole()) {
        return;
    }
    ISketchPtr hole_sketch = this->model_container->Sketchs->Add();
    hole_sketch->Name = _T("Профиль отверстия СМП");
    hole_sketch->Plane = this->base_plane;
    IFragmentDocumentPtr sketch_document = hole_sketch->BeginEdit();
    IDrawingContainerPtr drawing_container = this->getDrawingContainer
```

```
      (sketch_document);
    ICirclePtr figure = drawing_container->Circles->Add();
    figure->Style = ksCurveStyleEnum::ksCSNormal;
    figure->Xc = this->coordinates.X;
    figure->Yc = this->coordinates.Y;
    figure->Radius = this->holeRadius;
    figure->Update();
    hole_sketch->EndEdit();

    ICutExtrusionPtr cut_extrusion = this->model_container->Extrusions->Add(ExtrusionType: ksObj3dTypeEnum::o3d_cutExtrusion);
    cut_extrusion->Name = _T("Отверстие СМП");
    cut_extrusion->SetSideParameters(
        Normal: false, ExtrusionType: ksEndTypeEnum::etThroughAll,
        Depth: this->getHeight(), DraftValue: 0, DraftOutward: true, DepthObject: NULL
    );
    cut_extrusion->Sketch = hole_sketch;
    cut_extrusion->Update();
}
```