**一、中机国际贴面工具的主要代码 （470+100+130+200 = 800行）**

**1．MpFaceOptions.xaml 470**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Runtime.InteropServices;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Forms;

using System.Windows.Input;

using System.Windows.Interop;

using System.Windows.Media;

using Autodesk.Revit.DB;

using Autodesk.Revit.UI;

using FaceWall.DockablePane;

using Color = System.Windows.Media.Color;

using Cursors = System.Windows.Input.Cursors;

using Form = System.Windows.Forms.Form;

using MessageBox = System.Windows.Forms.MessageBox;

namespace FaceWall

{

/// <summary>

/// UC1.xaml 的交互逻辑

/// </summary>

public partial class MpFaceOptions : IDockablePaneProvider

{

#region --- Properties

public static readonly DockablePaneId DockablePaneId\_FaceWall = new DockablePaneId(new Guid("F07A471F-6003-445C-8D51-3F2C63416891"));

private static UIApplication \_uiApp;

private UIControlledApplication \_uiControlledApp;

private bool \_isRegistered = false;

// 外部事件

private Autodesk.Revit.UI.ExternalEvent m\_exEvent;

private ExEventHandler m\_handler;

#endregion

#region --- 构造函数与初始化

private static MpFaceOptions \_faceOptionsPanel;

/// <summary>

/// Panel 的激活（全局中只有此一个Panel）

/// </summary>

/// <returns></returns>

public static MpFaceOptions UniqueObject(UIControlledApplication uiControledApp)

{

//

if (\_faceOptionsPanel != null) return \_faceOptionsPanel;

//

\_faceOptionsPanel = new MpFaceOptions();

return \_faceOptionsPanel;

}

/// <summary>

/// Panel 的激活（全局中只有此一个Panel）

/// </summary>

/// <returns></returns>

public static MpFaceOptions UniqueObject(UIApplication uiApp)

{

\_uiApp = uiApp;

//

if (\_faceOptionsPanel != null) return \_faceOptionsPanel;

//

\_faceOptionsPanel = new MpFaceOptions();

return \_faceOptionsPanel;

}

/// <summary>

/// 构造函数

/// </summary>

private MpFaceOptions()

{

InitializeComponent();

//

m\_handler = new ExEventHandler(this);

m\_exEvent = ExternalEvent.Create(m\_handler);

// 面层在Revit中的类别信息

var itemSource = new FaceCategoryMapping[]

{

new FaceCategoryMapping("墙",FaceFilter. CategoryIds[0]),

new FaceCategoryMapping("柱",FaceFilter. CategoryIds[1]),

new FaceCategoryMapping("楼板",FaceFilter. CategoryIds[2]),

new FaceCategoryMapping("结构框架",FaceFilter. CategoryIds[3]),

new FaceCategoryMapping("屋顶",FaceFilter. CategoryIds[4]),

new FaceCategoryMapping("天花板",FaceFilter. CategoryIds[5]),

};

ComboxCategory.ItemsSource = itemSource;

ComboxCategory.DisplayMemberPath = "Text";

ComboxCategory.SelectedValuePath = "CategoryId";

ComboxCategory.SelectedIndex = 0;

// 面层的类型

FaceFilter ff = new FaceFilter(\_uiApp.ActiveUIDocument.Document);

IList<string> types = ff.GetFaceTypes(ff.GetAllInDoc());

foreach (string t in types)

{

ComboxType.Items.Add(t);

}

ComboxType.SelectedIndex = 0;

// Revit主窗口的句柄

rvtHwnd = Process.GetCurrentProcess().MainWindowHandle; // 当 Revit中有其他的弹出窗口时，MainWindowHandle属性所指的可能就不是Revit程序的那个大窗口了。

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

// 设置窗口界面

AsWindows();

}

/// <summary>

/// Revit的主窗口的句柄。

/// 当 Revit中有其他的弹出窗口时，MainWindowHandle属性所指的可能就不是Revit程序的那个大窗口了。

/// </summary>

IntPtr rvtHwnd;

[DllImport("user32.dll")]

private static extern int SetWindowLong(IntPtr hWnd, int nIndex, IntPtr dwNewLong);

/// <summary>

/// 当此界面作为一个单独窗口时的设置

/// </summary>

private void AsWindows()

{

Window window = null;

window = this as Window;

window.Title = "面层选项";

window.WindowStartupLocation = WindowStartupLocation.CenterScreen;

window.ResizeMode = ResizeMode.NoResize;

// window.Topmost = true;

// 将窗口显示在Revit窗口之上

WindowInteropHelper wndHelper = new WindowInteropHelper(this);

IntPtr wpfHwnd = wndHelper.Handle;

//string title = string.Format("Autodesk Revit 2016 - [三维视图: {三维} - 绘制面层.rvt]",); // Autodesk Revit 2016 - [三维视图: {三维} - 绘制面层.rvt]

//rvtHwnd = Windows.FindWindow(null, "Autodesk Revit 2016 - [三维视图: {三维} - 绘制面层.rvt]");

SetWindowLong(wpfHwnd, -8, rvtHwnd);

}

/// <summary>

/// 当此界面作为Revit中的Dockable Pane时的设置

/// </summary>

private void AsPage()

{

Page page = null;

// page = this as Page;

}

#endregion

#region --- 界面的 禁用 与 启用、窗口的显示与隐藏

public void DozeOff()

{

EnableCommands(false);

}

public void WakeUp()

{

EnableCommands(true);

}

/// <summary>

///

/// </summary>

/// <param name="status"> True 表示启用，false 表示禁用 </param>

private void EnableCommands(bool status)

{

if (status == false)

{

this.Cursor = Cursors.Wait;

// 禁用子控件

foreach (UIElement uiE in GridContent.Children)

{

uiE.IsEnabled = false;

}

}

else

{

this.Cursor = Cursors.Arrow;

// 启用子控件

foreach (UIElement uiE in GridContent.Children)

{

uiE.IsEnabled = true;

}

}

}

private void Window\_KeyDown\_1(object sender, System.Windows.Input.KeyEventArgs e)

{

if (e.Key == Key.Escape)

{

Close();

}

}

private void Window\_Closing\_1(object sender, System.ComponentModel.CancelEventArgs e)

{

e.Cancel = true;

Hide();

}

private void Window\_Closed\_1(object sender, EventArgs e)

{

\_faceOptionsPanel = null;

}

#endregion

#region --- DockablePane 注册 与 显示 隐藏

public void RegisterPanel(UIApplication uiApp)

{

if (!\_isRegistered) // if (!\_isRegistered)

{

uiApp.RegisterDockablePane(DockablePaneId\_FaceWall, "面层参数", this);

\_isRegistered = true;

}

}

public void RegisterPanel(UIControlledApplication uiControledApp)

{

if (!\_isRegistered) // if (!\_isRegistered)

{

uiControledApp.RegisterDockablePane(DockablePaneId\_FaceWall, "面层参数", this);

\_isRegistered = true;

}

}

/// <summary>

/// 此函数由uiApp.RegisterDockablePane时自动被调用。

/// </summary>

/// <param name="data"> 通过给此 data 实例的属性赋值，来确定要注册的 DockablePane 的样式 </param>

void IDockablePaneProvider.SetupDockablePane(DockablePaneProviderData data)

{

data.FrameworkElement = (FrameworkElement)this;

data.InitialState = new Autodesk.Revit.UI.DockablePaneState();

data.InitialState.SetFloatingRectangle(new Autodesk.Revit.UI.Rectangle(100, 100, 200, 200));

data.InitialState.DockPosition = DockPosition.Floating;

}

public void ShowPanel(UIControlledApplication uiControlledApp)

{

\_uiControlledApp = uiControlledApp;

Autodesk.Revit.UI.DockablePane dp = uiControlledApp.GetDockablePane(DockablePaneId\_FaceWall);

dp.Show();

}

public void ShowPanel(UIApplication uiApp)

{

\_uiApp = uiApp;

Autodesk.Revit.UI.DockablePane dp = uiApp.GetDockablePane(DockablePaneId\_FaceWall);

dp.Show();

}

public void HidePanel(UIApplication uiApp)

{

Autodesk.Revit.UI.DockablePane dp = uiApp.GetDockablePane(DockablePaneId\_FaceWall);

dp.Hide();

}

public void HidePanel(UIControlledApplication uiApp)

{

Autodesk.Revit.UI.DockablePane dp = uiApp.GetDockablePane(DockablePaneId\_FaceWall);

dp.Hide();

}

public Autodesk.Revit.UI.DockablePane GetPanel(UIControlledApplication uiControlledApp)

{

\_uiControlledApp = uiControlledApp;

Autodesk.Revit.UI.DockablePane dp = uiControlledApp.GetDockablePane(DockablePaneId\_FaceWall);

return dp;

}

public Autodesk.Revit.UI.DockablePane GetPanel(UIApplication uiApp)

{

\_uiApp = uiApp;

Autodesk.Revit.UI.DockablePane dp = uiApp.GetDockablePane(DockablePaneId\_FaceWall);

return dp;

}

#endregion

#region --- 设置 颜色 、 厚度、面层类型

private void ColorBoard\_MouseDown(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

ColorDialog loColorForm = new ColorDialog

{

FullOpen = true,

};

System.Drawing.Color color;

Utils.ConvertColor(((SolidColorBrush)ColorBoard.Background).Color, out color);

loColorForm.Color = color;

if (loColorForm.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

Color clr = new Color()

{

R = loColorForm.Color.R,

G = loColorForm.Color.G,

B = loColorForm.Color.B,

A = loColorForm.Color.A

};

ColorBoard.Background = new SolidColorBrush(clr);

}

}

private void ColorBoard\_MouseEnter(object sender, System.Windows.Input.MouseEventArgs e)

{

ColorBoard.ToolTip = ColorBoard.Background.ToString();

}

private void TextBlockThickness\_TextChanged(object sender, System.Windows.Controls.TextChangedEventArgs e)

{

string str = TextBlockThickness.Text;

if (string.IsNullOrEmpty(str))

{

return;

}

if (str == ".")

{

TextBlockThickness.Text = "0.";

TextBlockThickness.SelectionStart = 2;

return;

}

double thickness = 0;

if (double.TryParse(str, out thickness) && thickness >= 0)

{

}

else

{

TextBlockThickness.Text = "";

}

}

// 面层类型设置

private void ButtonDelType\_Click(object sender, RoutedEventArgs e)

{

int index = ComboxType.SelectedIndex;

if (index >= 0)

{

ComboxType.Items.RemoveAt(index);

// 选择一项

ComboxType.SelectedIndex = (index < ComboxType.Items.Count) ? index : index - 1;

}

}

private void ButtonAddType\_Click(object sender, RoutedEventArgs e)

{

FormAddFaceType addFaceType = new FormAddFaceType();

if (addFaceType.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

if (!ComboxType.Items.Contains(addFaceType.FaceType))

{

ComboxType.Items.Add(addFaceType.FaceType);

ComboxType.SelectedIndex = ComboxType.Items.Count - 1;

}

}

}

#endregion

#region --- 执行操作 m\_exEvent.Raise()

private void ButtonDrawFace\_Click(object sender, RoutedEventArgs e)

{

m\_handler.RequestId = ModelessCommandId.DrawFace;

m\_handler.DrawFaceOptions = GetFaceOptions();

m\_exEvent.Raise();

DozeOff();

}

private void ButtonFilter\_Click(object sender, RoutedEventArgs e)

{

m\_handler.RequestId = ModelessCommandId.Filter;

m\_handler.DrawFaceOptions = GetFaceOptions();

m\_exEvent.Raise();

}

private void ButtonSelectAll\_Click(object sender, RoutedEventArgs e)

{

m\_handler.RequestId = ModelessCommandId.SelectAll;

m\_handler.DrawFaceOptions = GetFaceOptions();

m\_exEvent.Raise();

}

#endregion

#region --- 绘制面层选项 FaceOptions

/// <summary>

/// 从界面中获取面层的设置选项信息

/// </summary>

/// <returns></returns>

public FaceOptions GetFaceOptions()

{

FaceOptions op = new FaceOptions(includeSameNormal: CheckBoxSameFaces.IsChecked ?? false);

// 颜色

Autodesk.Revit.DB.Color c1;

Utils.ConvertColor(((SolidColorBrush)ColorBoard.Background).Color, out c1);

op.Color = c1;

// 厚度

double thickNess;

double.TryParse(TextBlockThickness.Text, out thickNess);

op.SurfaceThickness = thickNess / 1000;

// 类别

ElementId categoryId = ComboxCategory.SelectedValue as ElementId;

op.CategoryId = categoryId ?? new ElementId(BuiltInCategory.OST\_Walls);

// 面层类型

op.FaceType = ComboxType.Text;

return op;

}

private class FaceCategoryMapping

{

public FaceCategoryMapping(string text, ElementId categoryId)

{

Text = text;

CategoryId = categoryId;

}

public ElementId CategoryId { get; set; }

public string Text { get; set; }

}

#endregion

}

}

**ExEventHandler.cs 100**

using System;

using System.Linq;

using System.Windows.Forms;

using Autodesk.Revit.UI;

using Autodesk.Revit.UI.Events;

namespace FaceWall

{

class ExEventHandler : IExternalEventHandler

{

private MpFaceOptions \_mainPanel;

public FaceOptions DrawFaceOptions;

public ModelessCommandId RequestId;

public ExEventHandler(MpFaceOptions mainPanel)

{

\_mainPanel = mainPanel;

}

public string GetName()

{

return "绘制面层";

}

/// <summary>

/// Called to execute an API command and update the UI after the command is finished.

/// </summary>

public void Execute(UIApplication uiApp)

{

UIDocument uiDoc = uiApp.ActiveUIDocument;

try

{

switch (RequestId)

{

case ModelessCommandId.DrawFace:

// 绘制面层

FaceDrawer fd = new FaceDrawer();

fd.DrawFaces(uiDoc, DrawFaceOptions);

break;

case ModelessCommandId.Filter:

FaceFilter ff2 = new FaceFilter(uiDoc.Document);

var faces1 = ff2.FilterSelected(DrawFaceOptions);

//

uiDoc.Selection.SetElementIds(faces1);

break;

case ModelessCommandId.SelectAll:

FaceFilter ff3 = new FaceFilter(uiDoc.Document);

var faces2 = ff3.GetAllInSelected();

//

uiDoc.Selection.SetElementIds(faces2.Select(r => r.FaceElement.Id).ToList()); ;

break;

}

}

catch (Exception ex)

{

Utils.ShowErrorMessage(ex, RequestId.ToString(), "面层绘制或过滤");

}

finally

{

\_mainPanel.WakeUp();

}

}

}

public enum ModelessCommandId : int

{

/// <summary>

/// 根据设置信息绘制新的面层

/// </summary>

DrawFace,

/// <summary>

/// 过滤选择的单元集合或者整个文档中的面层单元

/// </summary>

SelectAll,

/// <summary>

/// 根据指定的类型等信息，来过滤选择的单元中的面层单元

/// </summary>

Filter

}

}

**WallFace.cs 130**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Autodesk.Revit.DB;

namespace FaceWall

{

public class WallFace

{

public readonly DirectShape FaceElement;

/// <summary>

/// 构造函数

/// </summary>

/// <param name="faceElement">请自行确保 faceElement 所对应的单元是一个面层对象</param>

private WallFace(DirectShape faceElement)

{

FaceElement = faceElement;

}

#region --- 提取参数

/// <summary>

/// 获取面层对象的标识符。正常情况下，此参数的值应该是一个常数“CMIE\_面层”

/// </summary>

/// <param name="ds"> </param>

/// <param name="wallFace"> 如果此函数返回 true，返回面层对象，如果此函数返回False，则此参数返回 null </param>

/// <returns>如果没有找到此参数，则返回false。</returns>

public static bool IsWallFace(Element ds, out WallFace wallFace)

{

if (!(ds is DirectShape))

{

wallFace = null;

return false;

}

return IsWallFace((DirectShape)ds, out wallFace);

}

/// <summary>

/// 获取面层对象的标识符。正常情况下，此参数的值应该是一个常数“CMIE\_面层”

/// </summary>

/// <param name="ds"> </param>

/// <param name="wallFace"> 如果此函数返回 true，返回面层对象，如果此函数返回False，则此参数返回 null </param>

/// <returns>如果没有找到此参数，则返回false。</returns>

public static bool IsWallFace(DirectShape ds, out WallFace wallFace)

{

if (ds == null)

{

wallFace = null;

return false;

}

Parameter pa = ds.get\_Parameter(GlobalParameters.sp\_FaceIdTag\_guid);

if (pa == null)

{

wallFace = null;

return false;

}

string idTag = pa.AsString();

if (idTag == null || !pa.AsString().Equals(GlobalParameters.FaceIdentificaion))

{

wallFace = null;

return false;

}

wallFace = new WallFace(ds);

return true;

}

/// <summary>

/// 获取面层类型，如“防水”

/// </summary>

/// <param name="faceType">提取到的面层类型信息</param>

/// <returns>如果没有找到此参数，则返回false。</returns>

public bool GetFaceType(out string faceType)

{

Parameter pa = FaceElement.get\_Parameter(GlobalParameters.sp\_FaceType\_guid);

if (pa == null)

{

faceType = null;

return false;

}

faceType = pa.AsString();

return true;

}

/// <summary>

/// 获取面层的面积

/// </summary>

/// <param name="area">提取到的面层面积信息</param>

/// <returns>如果没有找到此参数，则返回false。</returns>

public bool GetArea(out double area)

{

Parameter pa = FaceElement.get\_Parameter(GlobalParameters.sp\_Area\_guid);

if (pa == null)

{

area = -1;

return false;

}

area = pa.AsDouble();

return true;

}

/// <summary>

/// 获取面层的体积

/// </summary>

/// <param name="volumn">提取到的面层体积信息</param>

/// <returns>如果没有找到此参数，则返回false。</returns>

public bool GetVolumn(out double volumn)

{

Parameter pa = FaceElement.get\_Parameter(GlobalParameters.sp\_Volumn\_guid);

if (pa == null)

{

volumn = -1;

return false;

}

volumn = pa.AsDouble();

return true;

}

#endregion

}

}

**FaceFilter.cs 200**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Autodesk.Revit.DB;

using Autodesk.Revit.DB.Architecture;

using Autodesk.Revit.UI;

using Autodesk.Revit.UI.Selection;

namespace FaceWall

{

public class FaceFilter

{

/// <summary>

/// 所有的面层对象可能的类别的集合

/// </summary>

public static ElementId[] CategoryIds = new ElementId[]

{

new ElementId(BuiltInCategory.OST\_Walls),

new ElementId(BuiltInCategory.OST\_Columns),

new ElementId(BuiltInCategory.OST\_Floors),

new ElementId(BuiltInCategory.OST\_StructuralFraming),

new ElementId(BuiltInCategory.OST\_Roofs),

new ElementId(BuiltInCategory.OST\_Ceilings),

};

private Document \_doc;

UIDocument \_uiDoc;

/// <summary>

/// 构造函数

/// </summary>

/// <param name="doc"></param>

public FaceFilter(Document doc)

{

\_doc = doc;

\_uiDoc = new UIDocument(doc);

}

#region --- 过滤出集合中的面层对象

/// <summary>

/// 从整个文档中的单元集合中过滤出面层对象

/// </summary>

public ICollection<WallFace> GetAllInDoc()

{

ICollection<WallFace> faces = faceLookup(\_doc, null);

return faces;

}

/// <summary>

/// 从选择的单元集合或者整个文档中的单元集合中过滤出面层对象，并将其在界面中选中

/// </summary>

public ICollection<WallFace> GetAllInSelected()

{

ICollection<WallFace> faces;

//

ICollection<ElementId> eleIds = \_uiDoc.Selection.GetElementIds();

//

// 如果没有选择任何单元，则从整个文档中进行搜索

faces = eleIds.Count == 0

? faceLookup(\_doc, null)

: faceLookup(\_doc, eleIds);

//

return faces;

}

/// <summary>

/// 从选择的单元集合或者整个文档中的单元集合中，根据指定的面层选项来进行过滤

/// </summary>

public ICollection<ElementId> FilterSelected(FaceOptions filter)

{

ICollection<ElementId> faces;

//

ICollection<ElementId> eleIds = \_uiDoc.Selection.GetElementIds();

//

// 如果没有选择任何单元，则从整个文档中进行搜索

faces = eleIds.Count == 0

? Filterfaces(\_doc, null, filter)

: Filterfaces(\_doc, eleIds, filter);

//

return faces;

}

// 面层对象的选择

/// <summary>

/// 从整个文档中的指定集合中搜索面层对象

/// </summary>

/// <param name="doc"></param>

/// <param name="elementIds"> 如果其值为null，则表示搜索集合为整个文档中的所有单元 </param>

/// <returns></returns>

private IList<WallFace> faceLookup(Document doc, ICollection<ElementId> elementIds)

{

List<WallFace> faces = new List<WallFace>();

FilteredElementCollector coll;

coll = (elementIds == null) ? new FilteredElementCollector(doc) : new FilteredElementCollector(doc, elementIds);

// 首先判断类型

coll.OfClass(typeof(DirectShape));

// 判断单元的类别是否是指定的类别集合中的一个

List<Element> faceCategoryElems = coll.Where(ele => CategoryIds.Contains(ele.Category.Id)).ToList();

// 再判断参数中是否有标识参数

Parameter pa;

string tag;

foreach (Element ele in faceCategoryElems)

{

WallFace wf;

if (!WallFace.IsWallFace(ele, out wf))

{

continue;

}

// 满足所有过滤条件

faces.Add(wf);

}

return faces;

}

// 根据指定的类别与类型来进行面层对象的过滤

/// <summary>

/// 从整个文档中的指定集合中 按指定的过滤选项 搜索面层对象

/// </summary>

/// <param name="doc"></param>

/// <param name="elementIds"> 如果其值为null，则表示过滤集合为整个文档中的所有单元 </param>

/// <returns></returns>

private IList<ElementId> Filterfaces(Document doc, ICollection<ElementId> elementIds, FaceOptions opt)

{

List<ElementId> faces = new List<ElementId>();

FilteredElementCollector coll;

coll = (elementIds == null) ? new FilteredElementCollector(doc) : new FilteredElementCollector(doc, elementIds);

// 首先判断类型 与 过滤指定的类别

coll.OfClass(typeof(DirectShape)).OfCategoryId(opt.CategoryId);

// 跟据参数值进行过滤

string tag;

foreach (Element ele in coll)

{

WallFace wallface;

if (!WallFace.IsWallFace(ele as DirectShape, out wallface))

{

continue;

}

// 2. 过滤面层类型，如“防水”

bool hasType = wallface.GetFaceType(out tag);

if (!hasType || tag == null || !tag.Equals(opt.FaceType, StringComparison.OrdinalIgnoreCase))

{

continue;

}

// 满足所有过滤条件

faces.Add(ele.Id);

}

return faces;

}

#endregion

#region --- 根据参数值进行进一步过滤

/// <summary>

/// 从给出的面层对象集合中，获得所有的面层类型（比如涂料、防水等）

/// </summary>

/// <param name="faces"></param>

/// <returns> 返回的面层类型集合中，没有相同的项。 </returns>

public IList<string> GetFaceTypes(IEnumerable<WallFace> faces)

{

IList<string> types = new List<string>();

string type;

foreach (WallFace f in faces)

{

if (f.GetFaceType(out type) && !types.Contains(type))

{

types.Add(type);

}

}

return types;

}

#endregion

}

}