**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Старший преподаватель департамента программной инженерии факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Шершаков «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2019 г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2019 г. |

|  |  |
| --- | --- |
| ***Подп. и дата*** |  |
| ***Инв. № дубл.*** |  |
| ***Взам. инв. №*** |  |
| ***Подп. и дата*** |  |
| ***Инв. № подл*** | RU.17701729.04.01-01 ПЗ 01-1-ЛУ |

**ПРОГРАММА-РАСШИРЕНИЕ MICROSOFT VISIO ДЛЯ ИМПОРТА ГРАФОВ В ФОРМАТЕ DOT**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.04.01-01 12 01-1-ЛУ**

Исполнитель  
студент группы БПИ173  
\_\_\_\_\_\_\_\_\_\_ / Переплетчиков А. И. /  
«\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2019 г.

**Москва 2019**

УТВЕРЖДЕНRU.17701729.04.01-01 ПЗ 01-1-ЛУ

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл | RU.17701729.04.01-01 12 01-1 |

**ПРОГРАММА-РАСШИРЕНИЕ MICROSOFT VISIO ДЛЯ ИМПОРТА ГРАФОВ В ФОРМАТЕ DOT**

**Текст программы**

**RU.17701729.04.01-01 12 01-1**

**Листов 28**

**Москва 2019**

**Содержание**

[**1. Текст программы** 3](#_Toc8721311)

[**1.1. Класс ThisAddIn** 3](#_Toc8721312)

[**1.2. Класс Ribbon** 7](#_Toc8721313)

[**1.3. Класс GraphParser** 9](#_Toc8721314)

[**1.4. Класс VisioGraph** 9](#_Toc8721315)

[**1.2. Класс VisioColor** 19](#_Toc8721316)

# **1. Текст программы**

## **1.1. Класс ThisAddIn**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Xml.Linq;

using System.Windows.Forms;

using Visio = Microsoft.Office.Interop.Visio;

using Office = Microsoft.Office.Core;

using GraphLibrary;

namespace VisioAddIn

{

public partial class ThisAddIn

{

private Dictionary<Visio.Page, VisioGraph> graphs = new Dictionary<Visio.Page, VisioGraph>();

private void ThisAddIn\_Startup(object sender, System.EventArgs e)

{

}

private void ThisAddIn\_Shutdown(object sender, System.EventArgs e)

{

}

/// <summary>

/// Метод отображения графа в Visio

/// </summary>

/// <param name="input"></param>

public void ShowGraph(string input)

{

Application.ActiveDocument.Pages.BeforePageDelete += Globals.ThisAddIn.DeleteGraph;

Visio.Documents visioDocs = Application.Documents;

Visio.Page visioPage = Application.ActiveDocument.Pages.Add();

graphs.Add(visioPage, new VisioGraph(input));

graphs[visioPage].PresentGraphInVisio(visioDocs, visioPage);

Application.ActivePage.BeforeShapeDelete += DeleteShape;

Application.ActivePage.ConnectionsDeleted += DeleteEdge;

Application.ActivePage.ConnectionsAdded += AddEdge;

Application.ActivePage.TextChanged += ChangeText;

Application.ActivePage.ShapeAdded += AddShape;

}

/// <summary>

/// Метод добавления фигуры

/// </summary>

/// <param name="Shape"></param>

private void AddShape(Visio.Shape Shape)

{

if (graphs.ContainsKey(Application.ActivePage))

{

try

{

graphs[Application.ActivePage].AddNode(Shape);

}

catch (Exception exc)

{

ErrorMessage("Во время добавления вершины возникла следующая ошибка:\n" + exc.Message, "Не удалось добавить вершину");

}

}

}

/// <summary>

/// Метод, вызывающий при попытке изменить ребро

/// </summary>

public void Invert()

{

if (graphs.ContainsKey(Application.ActivePage))

{

try

{

graphs[Application.ActivePage].Invert(Application.ActiveWindow);

}

catch (Exception exc)

{

ErrorMessage("Во время инвертирования ребра возникла следующая ошибка:\n" + exc.Message, "Не удалось инвертировать ребро");

}

}

}

/// <summary>

/// Лэйаутинг

/// </summary>

public void Layout()

{

if (graphs.ContainsKey(Application.ActivePage))

{

try

{

Application.ActivePage.Layout();

}

catch (Exception exc)

{

ErrorMessage("Во время планировки возникла следующая ошибка:\n" + exc.Message, "Не удалось исправить планировку");

}

}

else

{

ErrorMessage("На данной странице отсутствует граф!", "Не удалось исправить планировку");

}

}

/// <summary>

/// Метод выделения вершин

/// </summary>

/// <param name="key"></param>

public void Select(int key)

{

if (graphs.ContainsKey(Application.ActivePage))

{

try

{

graphs[Application.ActivePage].Select(key, Application.ActiveWindow);

}

catch (Exception exc)

{

ErrorMessage("Во время выделения вершин возникла следующая ошибка:\n" + exc.Message, "Не удалось выделить вершины");

}

}

}

/// <summary>

/// Метод изменения текста вершины

/// </summary>

/// <param name="Shape"></param>

private void ChangeText(Visio.Shape Shape)

{

if (graphs.ContainsKey(Application.ActivePage))

{

try

{

graphs[Application.ActivePage].ChangeLabel(Shape);

}

catch (Exception exc)

{

ErrorMessage("Во время изменения текста возникла следующая ошибка:\n" + exc.Message, "Не удалось изменить текст");

}

}

}

/// <summary>

/// Метод разрыва соединения вершин

/// </summary>

/// <param name="Connects"></param>

private void DeleteEdge(Visio.Connects Connects)

{

if (graphs.ContainsKey(Application.ActivePage))

{

try

{

graphs[Application.ActivePage].DeleteEdge(Connects);

}

catch (Exception exc)

{

ErrorMessage("Во время удаления ребра возникла следующая ошибка:\n" + exc.Message, "Не удалось удалить ребро");

}

}

}

/// <summary>

/// Метод удаления фигуры

/// </summary>

/// <param name="Shape"></param>

private void DeleteShape(Visio.Shape Shape)

{

if (graphs.ContainsKey(Application.ActivePage))

{

try

{

graphs[Application.ActivePage].DeleteShape(Shape);

}

catch (Exception exc)

{

ErrorMessage("Во время удаления объекта возникла следующая ошибка:\n" + exc.Message, "Не удалось удалить объект");

}

}

}

/// <summary>

/// Метод удаления ребра

/// </summary>

/// <param name="Connects"></param>

private void AddEdge(Visio.Connects Connects)

{

if (graphs.ContainsKey(Application.ActivePage))

{

try

{

graphs[Application.ActivePage].AddEdge(Connects);

}

catch (Exception exc)

{

ErrorMessage("Во время добавления ребра возникла следующая ошибка:\n" + exc.Message, "Не удалосьдобавить ребро");

}

}

}

/// <summary>

/// Метод удаления страницы, в случае, если возникла ошибка

/// </summary>

public void RemovePageIfError()

{

DeleteGraph(Application.ActiveDocument.Pages[Application.ActiveDocument.Pages.Count]);

Application.ActiveDocument.Pages[Application.ActiveDocument.Pages.Count].Delete(1);

}

/// <summary>

/// Удаление графа из словаря, если была удалена страница

/// </summary>

/// <param name="Page"></param>

private void DeleteGraph(Visio.Page Page)

{

if (graphs.ContainsKey(Page))

{

graphs.Remove(Page);

}

}

/// <summary>

/// Метод экспорта графа в файл

/// </summary>

/// <param name="filePath"></param>

public void ExportGraph(string filePath)

{

if (graphs.ContainsKey(Application.ActivePage))

{

graphs[Application.ActivePage].ExportGraph(filePath);

}

else throw new ArgumentException("На данной странице не представлен граф");

}

/// <summary>

/// Метод для отображения сообщения об ошибке

/// </summary>

/// <param name="message"></param>

/// <param name="caption"></param>

public void ErrorMessage(string message, string caption)

{

MessageBoxButtons buttons = MessageBoxButtons.OK;

DialogResult result;

// Отобразить окошко об ошибке

result = MessageBox.Show(message, caption, buttons);

}

#region Код, автоматически созданный VSTO

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

private void InternalStartup()

{

this.Startup += new System.EventHandler(ThisAddIn\_Startup);

this.Shutdown += new System.EventHandler(ThisAddIn\_Shutdown);

}

#endregion

}

}

## **1.2. Класс Ribbon**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Windows.Forms;

using Microsoft.Office.Tools.Ribbon;

namespace VisioAddIn

{

public partial class Ribbon

{

private void Ribbon\_Load(object sender, RibbonUIEventArgs e)

{

}

private void openFileButton\_Click(object sender, RibbonControlEventArgs e)

{

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

string input = File.ReadAllText(openFileDialog.FileName);

try

{

Globals.ThisAddIn.ShowGraph(input);

}

catch (Exception exc)

{

Globals.ThisAddIn.RemovePageIfError();

Globals.ThisAddIn.ErrorMessage("Во время импорта графа возникла следующая ошибка:\n" + exc.Message, "Не удалось отобразить граф");

}

}

}

private void exportGraphButton\_Click(object sender, RibbonControlEventArgs e)

{

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

try

{

Globals.ThisAddIn.ExportGraph(saveFileDialog.FileName);

}

catch (Exception exc)

{

Globals.ThisAddIn.ErrorMessage("Во время экспорта графа возникла следующая ошибка:\n\n" + exc.Message, "Не удалось экспортировать граф");

}

}

}

private void selectAllNodesButton\_Click(object sender, RibbonControlEventArgs e)

{

Globals.ThisAddIn.Select(1);

}

private void selectConnectedNodeButton\_Click(object sender, RibbonControlEventArgs e)

{

Globals.ThisAddIn.Select(2);

}

private void selectNonConnectedNodesButton\_Click(object sender, RibbonControlEventArgs e)

{

Globals.ThisAddIn.Select(3);

}

private void selectEdgesButton\_Click(object sender, RibbonControlEventArgs e)

{

Globals.ThisAddIn.Select(4);

}

private void invertButton\_Click(object sender, RibbonControlEventArgs e)

{

Globals.ThisAddIn.Invert();

}

private void layoutButton\_Click(object sender, RibbonControlEventArgs e)

{

Globals.ThisAddIn.Layout();

}

}

}

## **1.3. Класс GraphParser**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Graphviz4Net.Dot.AntlrParser;

using Graphviz4Net.Dot;

namespace GraphLibrary

{

public class GraphParser

{

public GraphParser() { }

public DotGraph<string> ParseGraphData(string code)

{

var parser = AntlrParserAdapter<string>.GetParser();

var result = parser.Parse(code);

return result;

}

}

}

## **1.4. Класс VisioGraph**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Graphviz4Net.Dot;

using Graphviz4Net.Dot.AntlrParser;

using Visio = Microsoft.Office.Interop.Visio;

using Office = Microsoft.Office.Core;

using System.IO;

namespace GraphLibrary

{

/// <summary>

/// Класс, объединяющий библиотеку Graphviz4Net и объектную модель Visio

/// </summary>

public class VisioGraph

{

private GraphParser gParser = new GraphParser();

private DotGraph<string> graph;

private Dictionary<DotVertex<string>, Visio.Shape> vertices = new Dictionary<DotVertex<string>, Visio.Shape>();

private Dictionary<DotEdge<string>, Visio.Shape> edges = new Dictionary<DotEdge<string>, Visio.Shape>();

private Dictionary<Visio.Shape, List<Visio.Shape>> newEdges = new Dictionary<Visio.Shape, List<Visio.Shape>>();

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="input">Код графа</param>

public VisioGraph(string input)

{

graph = gParser.ParseGraphData(input);

}

/// <summary>

/// Процедура отображения графа в документе Visio

/// </summary>

/// <param name="visioDocs">Документы Visio</param>

/// <param name="visioPage">Текущая страница в Visio</param>

public void PresentGraphInVisio(Visio.Documents visioDocs, Visio.Page visioPage)

{

PresentNodes(visioDocs, visioPage);

PresentEdges(visioDocs, visioPage);

visioPage.Layout();

}

/// <summary>

/// Процедура отображения вершин графа в документе Visio

/// </summary>

/// <param name="visioDocs">Документы Visio</param>

/// <param name="visioPage">Текущая страница в Visio</param>

private void PresentNodes(Visio.Documents visioDocs, Visio.Page visioPage)

{

// Мастер-объект базовых фигур Visio

Dictionary<string, Visio.Master> visioMasters = GetMasterShapes(visioDocs);

// Расстановка вершин графа

for (int i = 0; i < graph.AllVertices.Count(); ++i)

{

// Вершина

var node = graph.AllVertices.ElementAt(i);

// Стили вершины

string shape = node.Attributes.ContainsKey("shape") ? node.Attributes["shape"] : "ELLIPSE";

string label = node.Attributes.ContainsKey("label") ? node.Attributes["label"] : node.Id;

string color = node.Attributes.ContainsKey("color") ? node.Attributes["color"] : "black";

string fontcolor = node.Attributes.ContainsKey("fontcolor") ? node.Attributes["fontcolor"] : "black";

string style = node.Attributes.ContainsKey("style") ? node.Attributes["style"] == "filled" ? "filled" : LineStyle(node.Attributes["style"].ToLower()) : "1";

// Добавление вершины на страницу Visio

vertices.Add(node, visioPage.Drop(visioMasters[shape.ToUpper()], 1 + i / 2.0, 11 - i / 2.0));

// Установка стилей для фигуры на странице Visio

vertices[node].Text = label;

vertices[node].get\_CellsSRC((short)Visio.VisSectionIndices.visSectionCharacter, (short)Visio.VisRowIndices.visRowCharacter, (short)Visio.VisCellIndices.visCharacterColor).FormulaU = VisioColor.ColorToRgb(fontcolor.ToLower());

if (style == "filled")

vertices[node].get\_CellsSRC((short)Visio.VisSectionIndices.visSectionObject, (short)Visio.VisRowIndices.visRowFill, (short)Visio.VisCellIndices.visFillForegnd).FormulaU = VisioColor.ColorToRgb(color.ToLower());

else

vertices[node].get\_CellsU("LinePattern").FormulaU = style;

vertices[node].get\_CellsU("LineColor").FormulaU = VisioColor.ColorToRgb(color.ToLower());

// Ресайзинг

vertices[node].Resize(Visio.VisResizeDirection.visResizeDirNW, -0.8, Visio.VisUnitCodes.visInches);

// Чтобы не допустить пересечений

vertices[node].get\_CellsSRC((short)Visio.VisSectionIndices.visSectionObject, (short)Visio.VisRowIndices.visRowMisc, (short)Visio.VisCellIndices.visLOFlags).FormulaU = "1";

vertices[node].get\_CellsSRC((short)Visio.VisSectionIndices.visSectionObject, (short)Visio.VisRowIndices.visRowShapeLayout, (short)Visio.VisCellIndices.visSLOPlowCode).FormulaU = "2";

}

}

/// <summary>

/// Процедура отображения ребер графа в документе Visio

/// </summary>

/// <param name="visioDocs">Документы Visio</param>

/// <param name="visioPage">Текущая страница в Visio</param>

private void PresentEdges(Visio.Documents visioDocs, Visio.Page visioPage)

{

Visio.Document visioConnectors = visioDocs.OpenEx("Basic Flowchart Shapes (US units).vss", (short)Visio.VisOpenSaveArgs.visOpenDocked);

// Соединение вершин графа ребрами

for (int i = 0; i < graph.VerticesEdges.Count(); ++i)

{

// Ребро

var edge = graph.VerticesEdges.ElementAt(i);

// Фигура соединидельной линии (коннектора)

Visio.Shape connector = visioPage.Drop(visioConnectors.Masters.get\_ItemU("Dynamic connector"), 0, 0);

connector.get\_Cells("ConLineRouteExt").FormulaU = "2";

connector.get\_Cells("EndArrow").Formula = "=5";

// Стили ребра

string label = edge.Attributes.ContainsKey("label") ? edge.Attributes["label"] : "";

string color = edge.Attributes.ContainsKey("color") ? edge.Attributes["color"] : "black";

string linestyle = edge.Attributes.ContainsKey("style") ? LineStyle(edge.Attributes["style"].ToLower()) : "1";

// Установка стилей для фигуры на странице Visio

connector.Text = label;

connector.get\_CellsU("LineColor").FormulaU = VisioColor.ColorToRgb(color.ToLower());

connector.get\_CellsU("LinePattern").FormulaU = linestyle;

// Соединение вершин при помощи данного коннектора

vertices[edge.Source].AutoConnect(vertices[edge.Destination], Visio.VisAutoConnectDir.visAutoConnectDirDown, connector);

edges.Add(edge, vertices[edge.Source].FromConnects[vertices[edge.Source].FromConnects.Count].FromSheet);

// Удаление коннектора-болванки

connector.Delete();

}

}

/// <summary>

/// Метод инвертирования ребра

/// </summary>

/// <param name="window"></param>

public void Invert(Visio.Window window)

{

if (window != null)

{

// Для всех выделенных фигур

foreach (Visio.Shape shape in window.Selection)

{

// Проверяем, что это ребро

if (edges.ContainsValue(shape))

{

foreach (var edge in edges)

{

if (edge.Value == shape)

{

// Инвертируем ребро

Visio.Cell beginXCell = shape.get\_CellsSRC((short)Visio.VisSectionIndices.visSectionObject, (short)Visio.VisRowIndices.visRowXForm1D, (short)Visio.VisCellIndices.vis1DBeginX);

beginXCell.GlueTo(vertices[edge.Key.Destination].get\_CellsSRC((short)Visio.VisSectionIndices.visSectionObject, (short)Visio.VisRowIndices.visRowXFormOut, (short)Visio.VisCellIndices.visXFormPinX));

Visio.Cell endXCell = shape.get\_CellsSRC((short)Visio.VisSectionIndices.visSectionObject, (short)Visio.VisRowIndices.visRowXForm1D, (short)Visio.VisCellIndices.vis1DEndX);

endXCell.GlueTo(vertices[edge.Key.Source].get\_CellsSRC((short)Visio.VisSectionIndices.visSectionObject, (short)Visio.VisRowIndices.visRowXFormOut, (short)Visio.VisCellIndices.visXFormPinX));

// Заменяем старое ребро новым инвертированным

DotEdge<string> invertedEdge = new DotEdge<string>(edge.Key.Destination, edge.Key.Source, edge.Key.Attributes);

break;

}

}

}

else

{

throw new ArgumentException("Допустимо только инвертирование ребер!");

}

}

}

}

/// <summary>

/// Метод выделения вершин

/// </summary>

/// <param name="key"></param>

/// <param name="window"></param>

public void Select(int key, Visio.Window window)

{

if (window != null)

{

window.DeselectAll();

switch (key)

{

// Выделить все вершины

case 1:

foreach (var node in vertices)

{

window.Select(node.Value, 2);

}

break;

// Выделить все соединенные вершины

case 2:

foreach (var node in vertices)

{

if (node.Value.ConnectedShapes(Visio.VisConnectedShapesFlags.visConnectedShapesAllNodes, "").Length != 0)

{

window.Select(node.Value, 2);

}

}

break;

// Выделить все несоединенные вершины

case 3:

foreach (var node in vertices)

{

if (node.Value.ConnectedShapes(Visio.VisConnectedShapesFlags.visConnectedShapesAllNodes, "").Length == 0)

{

window.Select(node.Value, 2);

}

}

break;

// Выделить все ребра

case 4:

foreach (var edge in edges)

{

window.Select(edge.Value, 2);

}

break;

default:

throw new ArgumentException("Ключ не соответствует допустимому диапозону");

}

}

}

/// <summary>

/// Метод, вызывающийся при добавлении вершины

/// </summary>

/// <param name="shape"></param>

public void AddNode(Visio.Shape shape)

{

// Проверяем, что это соединительная линия

if (shape.Master.NameU == "Dynamic connector")

{

// Do nothing

}

else

{

// Если новая фигура допустимой формы

if (GetDotShapes().ContainsKey(shape.Master.NameU.ToUpper()))

{

if (shape.Text == "")

{

shape.Text = shape.GetHashCode().ToString();

}

DotVertex<string> newNode = new DotVertex<string>(shape.GetHashCode().ToString());

newNode.Attributes.Add("shape", GetDotShapes()[shape.Master.NameU.ToUpper()]);

newNode.Attributes.Add("label", shape.Text);

graph.AddVertex(newNode);

vertices.Add(newNode, shape);

}

else

{

throw new ArgumentException("Фигура не поддерживается языком DOT! Она не будет включена в список вершин графа!");

}

}

}

/// <summary>

/// Метод, вызывающийся при изменении текста фигуры

/// </summary>

/// <param name="shape"></param>

public void ChangeLabel(Visio.Shape shape)

{

// Если изменен текст вершины

if (vertices.ContainsValue(shape))

{

foreach (var node in vertices)

{

if (node.Value == shape)

{

if (node.Key.Attributes.ContainsKey("label"))

{

node.Key.Attributes["label"] = shape.Text;

}

else

{

node.Key.Attributes.Add("label", shape.Text);

}

break;

}

}

}

else

// Если изменен текст ребра

if (edges.ContainsValue(shape))

{

foreach (var edge in edges)

{

if (edge.Value == shape)

{

if (edge.Key.Attributes.ContainsKey("label"))

{

edge.Key.Attributes["label"] = shape.Text;

}

else

{

edge.Key.Attributes.Add("label", shape.Text);

}

break;

}

}

}

}

/// <summary>

/// Метод, вызывающийся при добавлении нового ребра между ребрами графа

/// </summary>

/// <param name="connects">Коннектор</param>

public void AddEdge(Visio.Connects connects)

{

// Если соединение добавлено с использованием существующей соединительной линии, т.е. добавляем вторую вершину

if (newEdges.ContainsKey(connects.FromSheet))

{

// Добавляем вторую вершину в пару связанных вершин

newEdges[connects.FromSheet].Add(connects.ToSheet);

// Если обе соединенные фигуры - это вершины нашего графа

if (vertices.ContainsValue(newEdges[connects.FromSheet][0]) && vertices.ContainsValue(newEdges[connects.FromSheet][1]))

{

DotVertex<string> from = new DotVertex<string>("");

DotVertex<string> to = new DotVertex<string>("");

// Ищем, каким вершинам они соответствуют

foreach (var node in vertices)

{

if (node.Value == newEdges[connects.FromSheet][0])

{

from = node.Key;

}

if (node.Value == newEdges[connects.FromSheet][1])

{

to = node.Key;

}

}

// Добавляем ребро с данными вершинами

DotEdge<string> edge = new DotEdge<string>(from, to);

graph.AddEdge(edge);

// Добавляем новое ребро в список ребер

edges.Add(edge, connects.FromSheet);

}

// Удаляем пару

newEdges.Remove(connects.FromSheet);

}

else

{

// Применяем стили к коннектору

connects.FromSheet.get\_Cells("ConLineRouteExt").FormulaU = "2";

connects.FromSheet.get\_Cells("EndArrow").Formula = "=5";

// Добавляем новый коннектор в словарь + вершину, от которой он стартовал, ожидая соединения ее со второй вершиной

List<Visio.Shape> listOfConnectingNodes = new List<Visio.Shape>();

listOfConnectingNodes.Add(connects.ToSheet);

newEdges.Add(connects.FromSheet, listOfConnectingNodes);

}

}

/// <summary>

/// Метод, вызывающийся при удалении ребра

/// </summary>

/// <param name="connects"></param>

public void DeleteEdge(Visio.Connects connects)

{

Visio.Shape shape = connects.FromSheet;

if (edges.ContainsValue(shape))

{

foreach (var edge in edges)

{

if (edge.Value == shape)

{

graph.RemoveEdge(edge.Key);

edges.Remove(edge.Key);

break;

}

}

}

}

/// <summary>

/// Метод, вызывающийся при удалении фигуры

/// </summary>

/// <param name="shape"></param>

public void DeleteShape(Visio.Shape shape)

{

// Если удалено ребро

if (edges.ContainsValue(shape))

{

Visio.Shape v1 = shape.Connects[1].ToSheet;

Visio.Shape v2 = shape.Connects[2].ToSheet;

if (vertices.ContainsValue(v1) && vertices.ContainsValue(v2))

{

foreach (var edge in edges)

{

if (edge.Value == shape)

{

graph.RemoveEdge(edge.Key);

edges.Remove(edge.Key);

break;

}

}

}

}

else

// Если удалена вершина

if (vertices.ContainsValue(shape))

{

foreach (var node in vertices)

{

if (node.Value == shape)

{

// Удаляем все ребра, смежные с данной вершиной

Dictionary<DotEdge<string>, Visio.Shape> edgesToDelete = new Dictionary<DotEdge<string>, Visio.Shape>();

foreach (var edge in edges)

{

if (edge.Key.Source == node.Key || edge.Key.Destination == node.Key)

{

graph.RemoveEdge(edge.Key);

edgesToDelete.Add(edge.Key, edge.Value);

}

}

foreach (var edge in edgesToDelete)

{

edges.Remove(edge.Key);

edge.Value.Delete();

}

// Удаляем вершину

graph.RemoveVertex(node.Key);

vertices.Remove(node.Key);

break;

}

}

}

}

/// <summary>

/// Процедура, сопоставляющая ключу-строке из допустимых фигур DOT мастер-фигуру Visio

/// </summary>

/// <param name="visioDocs"></param>

/// <returns>Словарь string-Visio.Master</returns>

private Dictionary<string, string> GetDotShapes()

{

Dictionary<string, string> result = new Dictionary<string, string>();

result.Add("TRIANGLE", "triangle");

result.Add("SQUARE", "square");

result.Add("HEXAGON", "hexagon");

result.Add("OCTAGON", "octagon");

result.Add("RECTANGLE", "rectangle");

result.Add("CIRCLE", "circle");

result.Add("ELLIPSE", "ellipse");

result.Add("DIAMOND", "diamond");

return result;

}

/// <summary>

/// Процедура, сопоставляющая ключу-строке из допустимых фигур DOT мастер-фигуру Visio

/// </summary>

/// <param name="visioDocs"></param>

/// <returns>Словарь string-Visio.Master</returns>

private Dictionary<string, Visio.Master> GetMasterShapes(Visio.Documents visioDocs)

{

Visio.Document visioStencil1 = visioDocs.OpenEx("Basic Shapes.vss", (short)Visio.VisOpenSaveArgs.visOpenDocked);

Visio.Document visioStencil2 = visioDocs.OpenEx("Audit Diagram Shapes.vss", (short)Visio.VisOpenSaveArgs.visOpenDocked);

Dictionary<string, Visio.Master> result = new Dictionary<string, Visio.Master>();

result.Add("TRIANGLE", visioStencil1.Masters.get\_ItemU(@"Triangle"));

result.Add("SQUARE", visioStencil1.Masters.get\_ItemU(@"Square"));

result.Add("PENTAGON", visioStencil1.Masters.get\_ItemU(@"Pentagon"));

result.Add("HEXAGON", visioStencil1.Masters.get\_ItemU(@"Hexagon"));

result.Add("OCTAGON", visioStencil1.Masters.get\_ItemU(@"Octagon"));

result.Add("RECTANGLE", visioStencil1.Masters.get\_ItemU(@"Rectangle"));

result.Add("RECT", visioStencil1.Masters.get\_ItemU(@"Rectangle"));

result.Add("BOX", visioStencil1.Masters.get\_ItemU(@"Rectangle"));

result.Add("CIRCLE", visioStencil1.Masters.get\_ItemU(@"Circle"));

result.Add("ELLIPSE", visioStencil1.Masters.get\_ItemU(@"Ellipse"));

result.Add("OVAL", visioStencil1.Masters.get\_ItemU(@"Ellipse"));

result.Add("DIAMOND", visioStencil1.Masters.get\_ItemU(@"Diamond"));

result.Add("PARALLELOGRAM", visioStencil2.Masters.get\_ItemU(@"I/O"));

return result;

}

/// <summary>

/// Процедура, возвращающая номер в формуле стиля линии

/// </summary>

/// <param name="styleString">Стиль в строковом формате</param>

/// <returns>Номер формулы</returns>

private string LineStyle(string styleString)

{

switch (styleString)

{

case "dashed":

return "2";

case "dotted":

return "3";

default:

return "1";

}

}

/// <summary>

/// Процедура экспорта графа в файл

/// </summary>

/// <param name="filePath">Путь к файлу</param>

public void ExportGraph(string filePath)

{

using (StreamWriter sw = new StreamWriter(filePath))

{

new GraphToDotConverter().Convert(sw, graph, new AttributesProvider());

}

}

}

public class AttributesProvider : IAttributesProvider

{

public IDictionary<string, string> GetVertexAttributes(object vertex)

{

return new Dictionary<string, string>();

}

}

}

## **1.5. Класс VisioColor**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GraphLibrary

{

static class VisioColor

{

public static string ColorToRgb(string color)

{

switch(color)

{

case "white":

return "RGB(255, 255, 255)";

case "red":

return "RGB(255,0,0)";

case "green":

return "RGB(0,255,0)";

case "aliceblue":

return "RGB(240, 248, 255)";

case "antiquewhite":

return "RGB(250, 235, 215)";

case "aquamarine":

return "RGB(127, 255, 212)";

case "azure":

return "RGB(240, 255, 255)";

case "beige":

return "RGB(245, 245, 220)";

case "black":

return "RGB(0,0,0)";

case "blue":

return "RGB(0,0,255)";

case "blueviolet":

return "RGB(138, 43, 226)";

case "brown":

return "RGB(165, 42, 42)";

case "burlywood":

return "RGB(222, 184, 135)";

case "cadetblue":

return "RGB(95, 158, 160)";

case "chartreuse":

return "RGB(127, 255, 0)";

case "chocolate":

return "RGB(210, 105, 30)";

case "coral":

return "RGB(255, 127, 80)";

case "cornflowerblue":

return "RGB(100, 149, 237)";

case "cornsilk":

return "RGB(255, 248, 220)";

case "crimson":

return "RGB(220, 20, 60)";

case "cyan":

return "RGB(0, 255, 255)";

case "darkgoldenrod":

return "RGB(184, 134, 11)";

case "darkgreen":

return "RGB(0, 100, 0)";

case "darkolivegreen":

return "RGB(85, 107, 47)";

case "darkorange":

return "RGB(255, 140, 0)";

case "darkorchild":

return "RGB(153, 50, 204)";

case "darksalmon":

return "RGB(233, 150, 122)";

case "darkseagreen":

return "RGB(143, 188, 143)";

case "darkslateblue":

return "RGB(72, 61, 139)";

case "darkslategray":

return "RGB(47, 79, 79)";

case "darkslategrey":

return "RGB(47, 79, 79)";

case "darkturquoise":

return "RGB(0, 206, 209)";

case "darkviolet":

return "RGB(148, 0, 211)";

case "deeppink":

return "RGB(255, 20, 147)";

case "deepskyblue":

return "RGB(0, 191, 255)";

case "dimgray":

return "RGB(105, 105, 105)";

case "dimgrey":

return "RGB(105, 105, 105)";

case "dodgerblue":

return "RGB(30, 144, 255)";

case "firebrick":

return "RGB(178, 34, 34)";

case "floralwhite":

return "RGB(255, 250, 240)";

case "forestgreen":

return "RGB(34, 139, 34)";

case "gainsboro":

return "RGB(220, 220, 200)";

case "ghostwhite":

return "RGB(248, 248, 255)";

case "gold":

return "RGB(255, 215, 0)";

case "goldenrod":

return "RGB(218, 165, 32)";

case "gray":

return "RGB(192, 192, 192)";

case "grey":

return "RGB(192, 192, 192)";

default:

throw new ArgumentException("Обнаружен недопустимый цвет");

}

}

}

}

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов | | | | Всего листов (страниц) в документе | № документа | Входя- щий № сопро- водит. докум. и дата | Под- пись | Дата |
| изменен- ных | заменен- ных | новых | аннулиро- ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |