# 2. Создание прототипа программы для выполнения динамических скриптов

Для изучения основ математического моделирования систем нам потребуется инструмент. Обучаться будем на примерах.



## 2.1. Введение в историю динамического выполнения кода

Идея динамической компиляции не является новой. Она возникла еще у пионеров программирования. Первым интерпретированным языком программирования высокого уровня был Lisp. Его интерпретатор был создан в 1958 году Стивом Расселом на компьютере IBM 704. Наконец-то эта идея захватила и мое воображение. Я решил написать программу для работы со скриптами, написанными на C#. Область применения – моделирование процессов и визуализация.

Программу назвал MathPanel – математическая панель. Нарисовал на бумаге минималистический интерфейс. Имеется 3 окна: команд, сообщений и графики (на рисунке скрыто, активируется при нажатии «График»).

Кнопки: «Выполнить» - осуществляет динамическую компиляцию и выполнения кода C# из окна команд.

«График» - переключается на графическое окно для визуалиции результатов. Рисование будем осуществлять с помощью объекта WebBrowser, javascript и canvas. Это медленнее, чем использование специализированных библиотек, но полезно для интеграции с веб-сайтом.

«Открыть» - поиск файла со скриптом и загрузка содержимого в окно команд.

«Сохранить» - содержимое окна команд сохраняется в файл.

«Компилировать» - на основе скрипта из окна команд создается DLL (динамическая библиотека).

«Новый скрипт» - в окно команд загружается шаблон скрипта.

«Картинка» - загрузка файла изображения и визуализация.

## 2.2. Разработка пользовательского интерфейса приложения MathPanel

Приступаем к реализации. В Visual Studio 2019 Community Edition создаем новое решение – WPF App (.NET Framework). Можно было бы и .NET Core (чтоб и под Linux), но для проверки идеи выбираем самый простой подход. Как говорится, сделай сначала просто.

Переходим к разработке прототипа. В файле MainWindow.xaml меняем

x:Class="WpfApp1.MainWindow"

на x:Class="WpfApp1.Dynamo"

В файле MainWindow.xaml.cs меняем

public partial class MainWindow : Window на

public partial class Dynamo : Window .

Сохраняем и запускаем проект. Там пока одно пустое окошко.

В графическом дизайнере среды создаем наш интерфейс, запускаем (рис. 2.1).

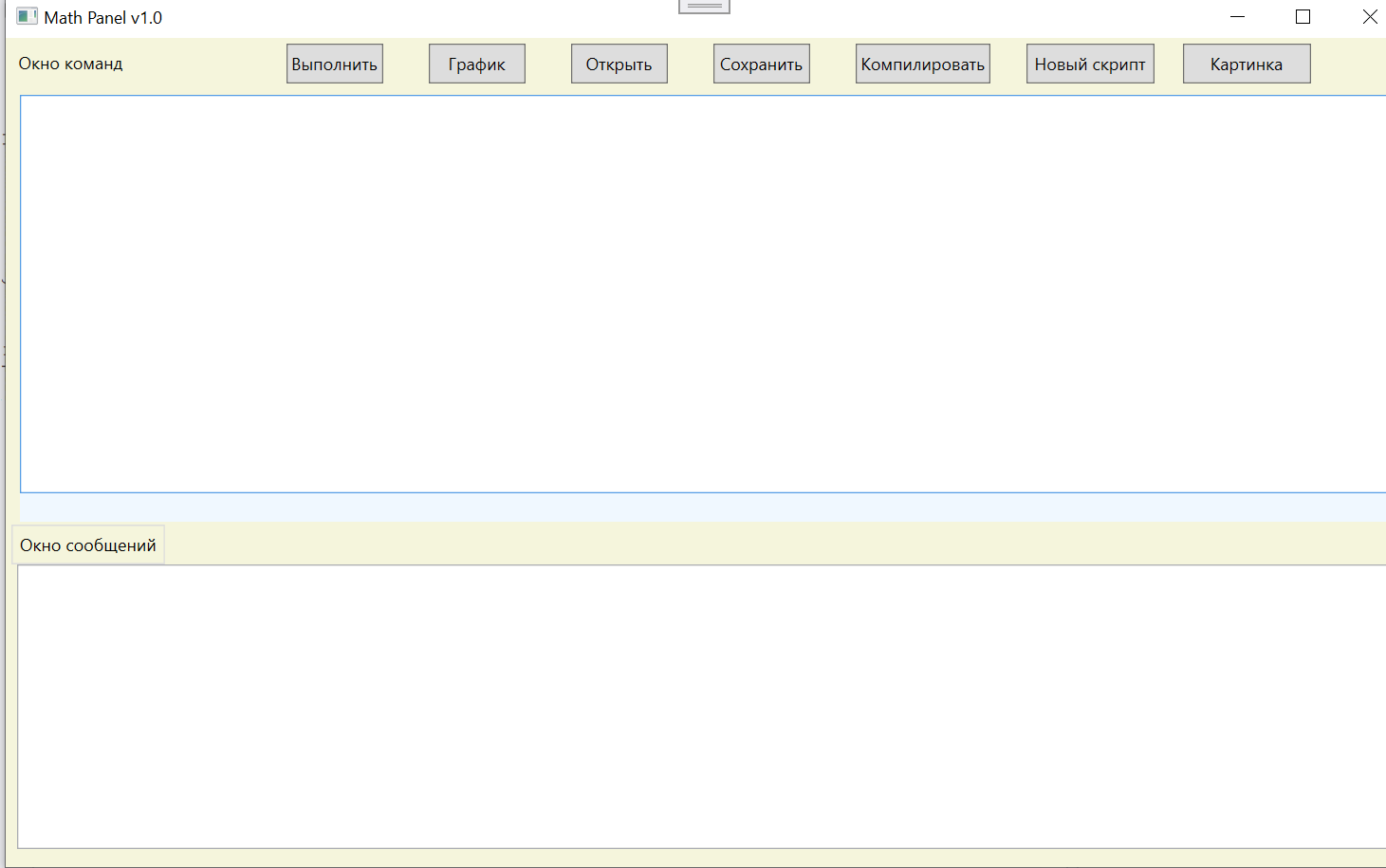


Рис. 2.1. Интерфейс MathPanel

Сам графический интерфейс задан в файле MainWindow.xaml и имеет следующий вид.

<Window x:Class="MathPanel.Dynamo"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:MathPanel"

mc:Ignorable="d"

Title="Math Panel" Height="620" Width="1200" Left="5" Top="5">

<Grid Margin="0,0,0,0" Height="600" Width="1200" Background="Beige">

<Label x:Name="Label1" Content="Команды" HorizontalAlignment="Left" Margin="4,4,0,0" VerticalAlignment="Top"/>

<Label x:Name="Label2" Content="Сообщения" HorizontalAlignment="Left" Margin="4,342,0,0" VerticalAlignment="Top" BorderThickness="1" BorderBrush="Gainsboro"/>

<Button x:Name="button1" Content="Выполнить" HorizontalAlignment="Left" Height="28" Margin="197,4,0,0" VerticalAlignment="Top" Width="68"/>

<Button x:Name="button2" Content="График" HorizontalAlignment="Left" Height="28" Margin="297,4,0,0" VerticalAlignment="Top" Width="68"/>

<Button x:Name="button3" Content="Открыть" HorizontalAlignment="Left" Height="28" Margin="397,4,0,0" VerticalAlignment="Top" Width="68"/>

<Button x:Name="button4" Content="Сохранить" HorizontalAlignment="Left" Height="28" Margin="497,4,0,0" VerticalAlignment="Top" Width="68"/>

<Button x:Name="button5" Content="Компилировать" HorizontalAlignment="Left" Height="28" Margin="597,4,0,0" VerticalAlignment="Top" Width="95"/>

<Button x:Name="button6" Content="Новый скрипт" HorizontalAlignment="Left" Height="28" Margin="717,4,0,0" VerticalAlignment="Top" Width="90"/>

<Button x:Name="button7" Content="Картинка" HorizontalAlignment="Left" Height="28" Margin="827,4,0,0" VerticalAlignment="Top" Width="90"/>

<ScrollViewer Name="sv1" VerticalScrollBarVisibility="Auto" BorderThickness="1" Width="1180" Margin="4,40,4,260" Height="320" Background="AliceBlue">

<TextBox x:Name="textBlock1" HorizontalAlignment="Left" VerticalScrollBarVisibility="Visible" TextWrapping="Wrap" AcceptsReturn="True" Text="" VerticalAlignment="Top" Height="280" Width="1150"/>

</ScrollViewer>

<TextBox x:Name="textBlock2" HorizontalAlignment="Left" VerticalScrollBarVisibility="Visible" TextWrapping="Wrap" AcceptsReturn="True" Text="" VerticalAlignment="Top" Height="200" Width="1150" Margin="8,370,42,0"/>

<WebBrowser x:Name="web1" HorizontalAlignment="Left" Height="550" VerticalAlignment="Bottom" Width="1180" Margin="0,0,0,9"/>

<Image x:Name="img1" HorizontalAlignment="Left" Height="550" VerticalAlignment="Bottom" Width="1180" Margin="0,0,0,9" />

</Grid>

</Window>

Элемент Window – корневой. В нем – Grid. В Grid заданы метки, кнопки, два текстовых блока, компоненты Webbrowser и Image. Код программы выглядит так.

namespace MathPanel

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class Dynamo : Window

{

public Dynamo()

{

InitializeComponent();

}

}

}

Приступаем к наращиванию мышц. Скроем 2 наших компонента в конструкторе класса.

public Dynamo()

{

InitializeComponent();

//скрыть веб-браузер и компонент картинки

web1.Visibility = Visibility.Hidden;

img1.Visibility = Visibility.Hidden;

}

После запуска видим, что появились два текстовых блока. Создадим обработчик для кнопки «Выполнить».

//обработчик кнопки "Выполнить"

private void Button1\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("Не готово!");

return;

}

И статическую переменную для версии программы

readonly static string version = " v1.0";

В конструктор добавим команды для модификации заголовка программы и назначение обработчика

this.Title += version;

button1.Click += Button1\_Click;

Запускаем программу, нажимаем на кнопку. Результат на рис.2.2.

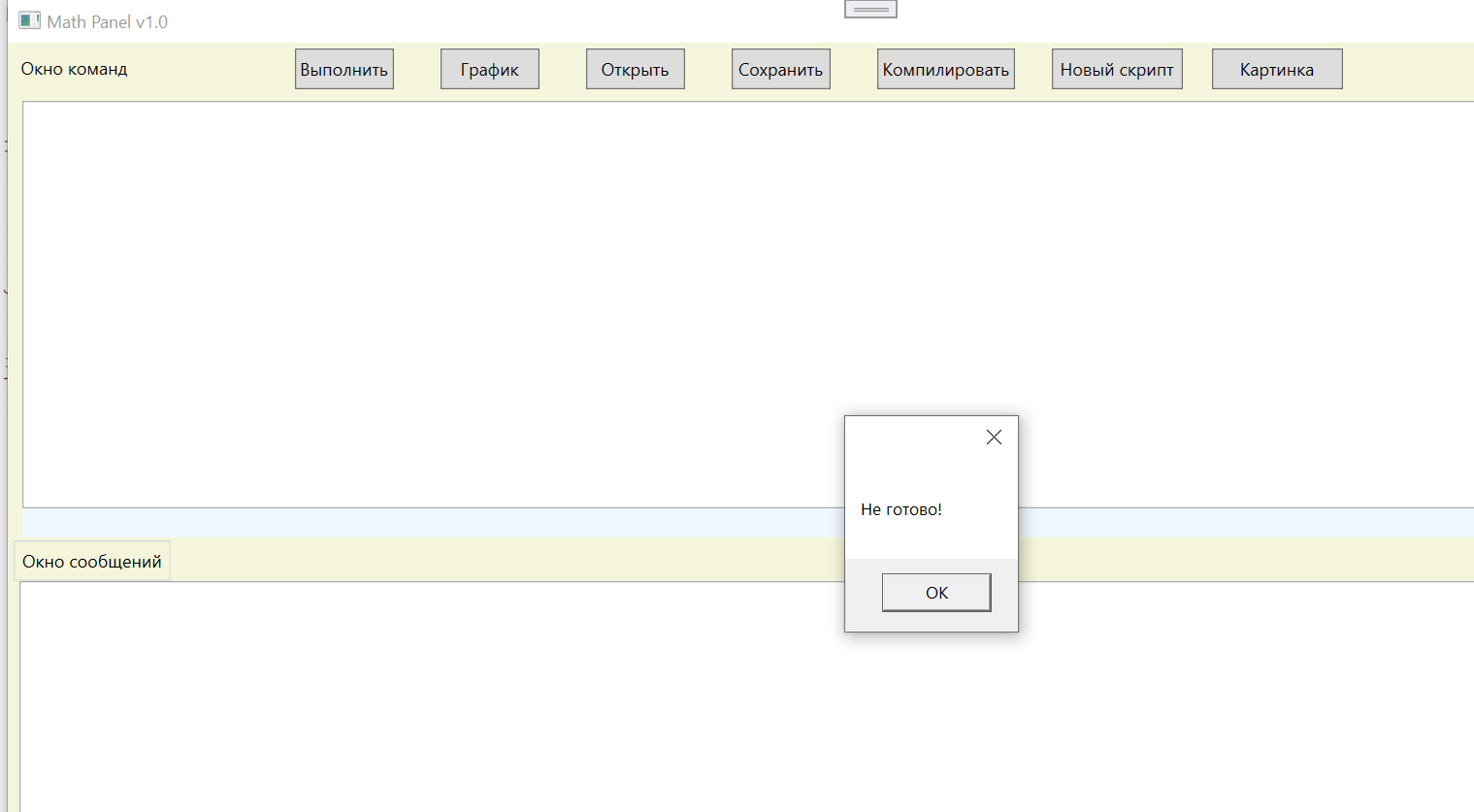


Рис.2.2. Первый результат

## 2.3. Методы ведения журнала и вывода в окно сообщений

Добавляем ссылку на библиотеку ввода-вывода

using System.IO;

название файла логирования

readonly static string sLogFile = "mathpanel.log";

метод для логирования в файл

/// <summary>

/// write to log-file

/// </summary>

public static void Log(string s)

{

try

{

StreamWriter sw = new StreamWriter(sLogFile, true, Encoding.UTF8);

sw.WriteLine(string.Format("{0:yyyy-MM-dd HH:mm:ss} {1}", DateTime.Now, s));

sw.Close();

}

catch (Exception se)

{

MessageBox.Show(se.ToString());

}

}

Здесь мы открываем поток для записи в файл с добавление к имеющемуся содержимому (второй параметр конструктора StreamWriter - true), записываем в него строку с добавленной меткой времени и сохраняем в файл. При работе с файлами возникают исключения. Для их перехвата используем try/catch блок.

В обработчик кнопки добавляем вызов

Log("Не готово");

Запускаем программу, нажимаем на кнопку, в текущей папке появился файл «mathpanel.log», в котором записано

2021-09-09 14:13:57 Не готово

Добавим еще один метод для вывода сообщений в нижнем окне программы. Мы планируем вызывать его не только из потока пользовательского интерфейса. Поэтому создаем новые переменные dispObj, bReady, txtConsole, txtCommand

static System.Windows.Threading.Dispatcher dispObj; //диспетчер UI-потока, через него обращения к элементам UI из других потоков

static bool bReady = true; //признак успешной инициализации

static TextBox txtConsole, txtCommand; //окна сообщений и комманд

Инициируем их в конструкторе

dispObj = Dispatcher;

txtCommand = textBlock1;

txtConsole = textBlock2;

bReady = true;

Создаем вспомогательный метод Console.

//запускаем циклические процессы в отдельном потоке

//чтобы обновлять интерфейс пользователя, используем запуск кода в UI

/// <summary>

/// добавить текст в окно сообщений

/// </summary>

/// <param name="s">текст</param>

public static void Console(string s, bool bNewLine = true)

{

//txtConsole.Text += (s + "\r\n");

if (!bReady || dispObj.HasShutdownStarted) return;

//мы запускаем код в UI потоке

dispObj.Invoke(delegate

{

if (bReady) txtConsole.Text += (s + (bNewLine ? "\r\n" : ""));

});

}

## 2.4. Динамическая компиляция с использованием C#

Приступаем к динамической компиляции. Потребуется информация о пути к системным библиотекам.

//путь к используемому фреймворку

static string frmPath = @"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.7.2\";

static string keyConsole = ""; //ввод с клавиатуры

static object dynClassInstance = null; //объект типа скрипт

static System.Threading.Thread my\_thread = null; //текущий поток со скриптом

readonly static List<System.Threading.Thread> lstThr = new List<Thread>(); //список всех потоков со скриптами

Добавляем пространства имен

//for dynamo

using System.CodeDom.Compiler;

using Microsoft.CSharp;

using System.Reflection;

using System.Threading;

Добавляем метод Process.

//компилировать и выполнить скрипт из окна "комманд"

public static void Process(string s)

{

//сборки

string[] includeAssemblies = { "MathPanel.exe",

frmPath + "System.dll",

frmPath + "System.Xaml.dll",

frmPath + "WindowsBase.dll",

frmPath + "PresentationFramework.dll",

frmPath + "PresentationCore.dll"

, frmPath + "System.Drawing.dll"

, frmPath + "System.Net.dll"

, frmPath + "System.Net.Http.dll"

};

//пространства имен

string[] includeNamespaces = { "MathPanel", "MathPanelExt", "System.Net.Sockets" };

keyConsole = "";

CompileDynamo(s, null, includeNamespaces, includeAssemblies);

//создаем новый поток

try

{

if (dynClassInstance != null)

{

my\_thread = new System.Threading.Thread(new System.Threading.ThreadStart(() => {

Type type = dynClassInstance.GetType();

MethodInfo methodInfo = type.GetMethod("Execute");

try

{

methodInfo.Invoke(dynClassInstance, null);

Dynamo.Console("Скрипт выполнен.");

}

catch (Exception yyy) { Dynamo.Console(yyy.ToString()); }

//my\_thread = null;

}));

my\_thread.Start();

lstThr.Add(my\_thread);

}

}

catch (Exception xxx) { Dynamo.Console(xxx.ToString()); }

}

И вспомогательный метод CompileDynamo

//создать объект типа скрипт

static object CompileDynamo(string code, Type outType = null, string[] includeNamespaces = null, string[] includeAssemblies = null)

{

StringBuilder namespaces = null;

object methodResult = null;

dynClassInstance = null;

using (CSharpCodeProvider codeProvider = new CSharpCodeProvider())

{

//ICodeCompiler codeCompiler = codeProvider.CreateCompiler();//obsolete!

CompilerParameters compileParams = new CompilerParameters

{

CompilerOptions = "/t:library",

GenerateInMemory = true

};

int ipos = code.IndexOf("///[DLL]");

if (ipos > 0)

{

int ipos2 = code.IndexOf("[/DLL]", ipos);

if (ipos2 > 0)

{

compileParams.ReferencedAssemblies.Add("MathPanel.exe");

string ass = code.Substring(ipos + 8, ipos2 - ipos - 8);

var arr = ass.Split(',');

foreach (string \_assembly in arr)

{

compileParams.ReferencedAssemblies.Add(frmPath + \_assembly.Trim());

}

}

}

else if (includeAssemblies != null && includeAssemblies.Length > 0)

{

foreach (string \_assembly in includeAssemblies)

{

compileParams.ReferencedAssemblies.Add(\_assembly);

}

}

if (includeNamespaces != null && includeNamespaces.Length > 0)

{

namespaces = new StringBuilder();

foreach (string \_namespace in includeNamespaces)

{

namespaces.Append(string.Format("using {0};\n", \_namespace));

}

}

if (code.IndexOf("namespace DynamoCode") < 0)

code = string.Format(

@"{1}

using System;

namespace DynamoCode{{

public class Script{{

public {2} Execute(){{

{3} {0};

}}

}}

}}",

code,

namespaces != null ? namespaces.ToString() : null,

outType != null ? outType.FullName : "void",

outType != null ? "return" : string.Empty

);

CompilerResults compileResult = codeProvider.CompileAssemblyFromSource(compileParams, code);

//codeCompiler.CompileAssemblyFromSource(compileParams, code);

if (compileResult.Errors.Count > 0)

{

//throw new Exception(compileResult.Errors[0].ErrorText);

Console("compile error: " + compileResult.Errors[0].ErrorText);

return "compile error: " + compileResult.Errors[0].ErrorText;

}

System.Reflection.Assembly assembly = compileResult.CompiledAssembly;

dynClassInstance = assembly.CreateInstance("DynamoCode.Script");

/\*Type type = dynClassInstance.GetType();

MethodInfo methodInfo = type.GetMethod("Execute");

methodResult = methodInfo.Invoke(dynClassInstance, null);\*/

}

return methodResult;

}

И меняем код обработчика кнопки

if (!bReady)

{

MessageBox.Show("Не готово!");

return;

}

Process(textBlock1.Text);

Запускаем программу, печатаем в окне команд

var hz = Math.Sqrt(3);

Dynamo.Console(hz.ToString());

Нажимаем «Выполнить», получаем 1,73205080756888 в окне сообщений.

Поясним детали. Обработчик кнопки считывает текст из окна команд и передает его в метод Process().

Process формирует список выполняемых модулей

string[] includeAssemblies

и список пространств имен

string[] includeNamespaces

Затем вызывает для компиляции кода метод CompileDynamo. В случае успеха

(dynClassInstance != null) есть истина.

Тогда создаем новый поток с делегатом, в котором запускается метод Execute из скомпилированного кода.

my\_thread = new System.Threading.Thread(new System.Threading.ThreadStart(() => {

Type type = dynClassInstance.GetType();

MethodInfo methodInfo = type.GetMethod("Execute");

try

{

methodInfo.Invoke(dynClassInstance, null);

Dynamo.Console("Done");

}

catch (Exception yyy) { Dynamo.Console(yyy.ToString()); }

}));

Стартуем поток и добавляем его в список (для последующей очистки).

my\_thread.Start();

lstThr.Add(my\_thread);

Собственно, компиляция происходит в методе CompileDynamo. Сначала создается объект codeProvider, который предоставляет доступ к экземплярам генератора и компилятора кода C#. Ему передаются параметры компиляции: создавать библиотеку в памяти.

using (CSharpCodeProvider codeProvider = new CSharpCodeProvider())

{

CompilerParameters compileParams = new CompilerParameters

{

CompilerOptions = "/t:library",

GenerateInMemory = true

};

Фрагмент позволяет указывать список нужных библиотек в компилируемом коде.

int ipos = code.IndexOf("///[DLL]");

if (ipos > 0)

{

int ipos2 = code.IndexOf("[/DLL]", ipos);

if (ipos2 > 0)

{

compileParams.ReferencedAssemblies.Add("MathPanel.exe");

string ass = code.Substring(ipos + 8, ipos2 - ipos - 8);

var arr = ass.Split(',');

foreach (string \_assembly in arr)

{

compileParams.ReferencedAssemblies.Add(frmPath + \_assembly.Trim());

}

}

}

Затем формируется окончательный код для компиляции. Если поставить точку прерывания, то можно зафиксировать код, который передается компилятору.

using MathPanel;

using MathPanelExt;

using System.Net.Sockets;

using System;

namespace DynamoCode{

public class Script{

public void Execute(){

**var hz = Math.Sqrt(3);**

**Dynamo.Console(hz.ToString());**

}

}

}

Таким образом происходит «обертывание» кода из окна команд в класс DynamoCode.Script с методом Execute.

Затем даем команду компилятору

CompilerResults compileResult = codeProvider.CompileAssemblyFromSource(compileParams, code);

В случае успеха создаем экземпляр сборки

if (compileResult.Errors.Count > 0)

{

Console("compile error: " + compileResult.Errors[0].ErrorText);

return "compile error: " + compileResult.Errors[0].ErrorText;

}

System.Reflection.Assembly assembly = compileResult.CompiledAssembly;

dynClassInstance = assembly.CreateInstance("DynamoCode.Script");

Теперь можно написать что-то посложнее для выполнения. Например, генерация чисел от 1 до 20 и потом снова до 1.

//for Excel increment cell values

for(int i = 1; i <= 20; i++)

Dynamo.Console(i.ToString());

for(int i = 19; i >= 1; i--)

Dynamo.Console(i.ToString());

Копируем результат из окна сообщений и вставляем в Эксель. Удобно.

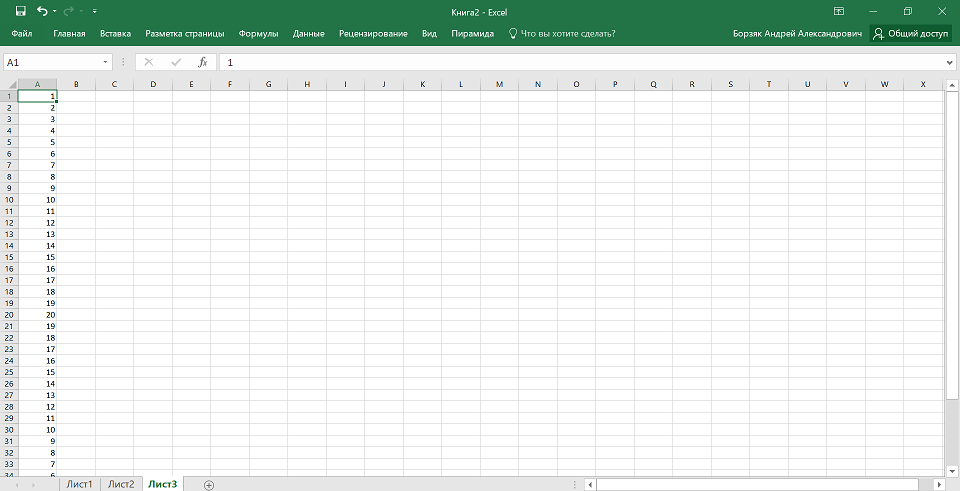


Рис.2.3. Копирование результата в Эксель

С реализацией первой кнопки «Выполнить» мы закончили. Переходим к другим кнопкам.

## 2.5. Кнопки «График», «Открыть», «Сохранить»

Кнопка «График» переключает на графическое окно для визуалиции результатов. Мы будем использовать html-страницу «test\_graph.htm» из текущей директории.

В ней следующее содержимое.

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

<title>Canvas drawing</title>

<style type="text/css">

body { background-color:#ffe; border:2px solid #cfc; }

</style>

<script type="text/javascript" src="jquery-1.11.1.min.js"></script>

</head>

<body>

<h1>Canvas drawing</h1>

</body>

</html>

В конструктор добавляем код загрузки.

string sDir = AppDomain.CurrentDomain.BaseDirectory;

web1.Navigate("file:///" + sDir + "test\_graph.htm");

button2.Click += Button2\_Click;

Добавляем новый обработчик

//скрыть/показать элементы интерфейса

private void Button2\_Click(object sender, RoutedEventArgs e)

{

Label1.Visibility = Label1.IsVisible ? Visibility.Hidden : Visibility.Visible;

Label2.Visibility = Label2.IsVisible ? Visibility.Hidden : Visibility.Visible;

sv1.Visibility = sv1.IsVisible ? Visibility.Hidden : Visibility.Visible;

textBlock2.Visibility = textBlock2.IsVisible ? Visibility.Hidden : Visibility.Visible;

web1.Visibility = web1.IsVisible ? Visibility.Hidden : Visibility.Visible;

button2.Content = Label1.IsVisible ? "График" : "Команды";

}

Здесь происходит переключение видимости. Если окно команд видимо, делаем его невидимым, а компонент WebBrowser видимым. И наоборот.

Запускаем программу, получаем после нажатия кнопки «График» переключение на html-страницу (рис.2.4).

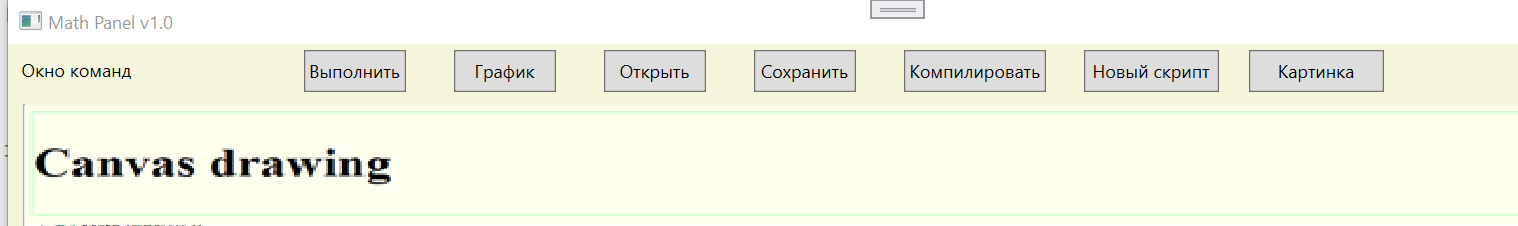


Рис.2.4. Результат нажатия кнопки «График»

С реализацией кнопки «График» закончено.

Кнопка «Готово». Добавляем пространство имен

//for open dialog

using Microsoft.Win32;

Добавляем обработчик

// загрузить скрипт из файла

private void Button3\_Click(object sender, RoutedEventArgs e)

{

string data;

OpenFileDialog openFileDialog = new OpenFileDialog();

if (openFileDialog.ShowDialog() == true)

{

data = File.ReadAllText(openFileDialog.FileName, Encoding.UTF8);

textBlock1.Text = data;

}

}

И ссылку на него в конструкторе

button3.Click += Button3\_Click;

Готово.

Кнопка « Сохранить ». Добавляем обработчик

// сохранить скрипт в файле

private void Button4\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

CheckFileExists = false

};

if (openFileDialog.ShowDialog() == true)

{

File.WriteAllText(openFileDialog.FileName, textBlock1.Text, Encoding.UTF8);

}

}

И ссылку на него в конструкторе

button4.Click += Button4\_Click;

Готово.

Кнопка «Компилировать». Добавляем обработчик

private void Button5\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

CheckFileExists = false

};

if (openFileDialog.ShowDialog() == false) return;

string fname = openFileDialog.FileName;

if (fname.LastIndexOf(".cs") == fname.Length - 3)

{

var data = File.ReadAllText(openFileDialog.FileName, Encoding.UTF8);

textBlock1.Text = data;

fname = fname.Substring(0, fname.Length - 3) + ".dll";

}

string sourceDll = textBlock1.Text;

string outputAssembly = fname;

CompilerResults results;

try

{

CSharpCodeProvider provider = new CSharpCodeProvider();

//признак build dll

CompilerParameters compilerParams2 = new CompilerParameters

{ OutputAssembly = outputAssembly, GenerateExecutable = false };

//to allow LINQ

compilerParams2.ReferencedAssemblies.Add("System.Core.Dll");

//Компиляция dll

results = provider.CompileAssemblyFromSource(compilerParams2, sourceDll);

//Выводим информацию об ошибках

Console(string.Format("Number of Errors DLL: {0}", results.Errors.Count));

foreach (CompilerError err in results.Errors)

{

Console(string.Format("ERROR {0}", err.ErrorText));

}

//test

Console(string.Format("Try Assembly:"));

Assembly assembly = Assembly.LoadFile(outputAssembly);

Type type = assembly.GetType("MathPanelExt.QuadroEqu");

MethodInfo method = type.GetMethod("Solve");

double x1 = 0, x2 = 0;

object[] pars = new object[5];

pars[0] = 1.0;

pars[1] = 0.0;

pars[2] = -1.0;

pars[3] = null;

pars[4] = null;

object res = method.Invoke(null, pars);

//bool blResult = (bool)res;

//if (blResult)

{

x1 = (double)pars[3];

x2 = (double)pars[4];

}

Console(string.Format("x1 {0}, x2 {1}", x1, x2));

//end of test

}

catch (Exception ex) { Console(ex.ToString()); }

}

И ссылку на него в конструкторе

button5.Click += Button5\_Click;

Создадим простой класс и сохраним его в папке scripts.

using System;

namespace MathPanelExt

{

/// <summary>

/// class for solving the quadratic equation

/// </summary>

public class QuadroEqu

{

public QuadroEqu()

{

}

/// <summary>

/// calculate roots of quadratic equation

/// </summary>

public static void Solve(double a, double b, double c, out double x1, out double x2)

{

double discr = Math.Sqrt(b \* b - 4 \* a \* c);

x1 = (-b - discr) / (2 \* a);

x2 = (-b + discr) / (2 \* a);

}

}

}

Запускаем программу, нажимаем на «Компилировать». Получаем результат в окне сообщений.

Try Assembly:

x1 -1, x2 1

После этого блок test можно закомментировать или удалить. Готово.

## 2.6. Шаблон для скриптов MathPanel

Кнопка «Новый скрипт». Создадим шаблон для создания скриптов в виде файла «template.cs» в папку scripts.

using MathPanel;

using MathPanelExt;

using System.Net.Sockets;

using System;

///сборки для добавления

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("Скрипт стартовал!");

//добавьте свой код ниже

}

}

}

Добавляем обработчик

// обработчик кнопки "Новый скрипт"

private void Button6\_Click(object sender, RoutedEventArgs e)

{

string data = File.ReadAllText("template.cs", Encoding.UTF8);

textBlock1.Text = data;

}

И ссылку на него в конструкторе

button6.Click += Button6\_Click;

Копируем шаблон в папку Debug (Release). Запускаем, нажимаем, получаем в окне сообщений:

Скрипт стартовал!

Скрипт выполнен.

Готово.

## 2.7. Обработчик кнопки «Изображение»

Кнопка «Картинка». Добавляем статическую переменную

static System.Windows.Controls.Image imgStatic;

Инициализируем ее в конструкторе

imgStatic = img1;

Добавляем обработчик

//load image

private void Button7\_Click(object sender, RoutedEventArgs e)

{

Label1.Visibility = Visibility.Hidden;

Label2.Visibility = Visibility.Hidden;

sv1.Visibility = Visibility.Hidden;

textBlock2.Visibility = Visibility.Hidden;

web1.Visibility = Visibility.Hidden;

img1.Visibility = Visibility.Visible;

button2.Content = "Команды";

string path;

OpenFileDialog openFileDialog = new OpenFileDialog();

if (openFileDialog.ShowDialog() == true)

{

path = openFileDialog.FileName;

var bitmap = new System.Windows.Media.Imaging.BitmapImage(new Uri(path, UriKind.Absolute));

img1.Source = bitmap;

}

}

И ссылку на него в конструкторе

button7.Click += Button7\_Click;

Запускаем приложение, нажимаем «Картинка»., выбираем картинку, получаем результат (рис.2.5).

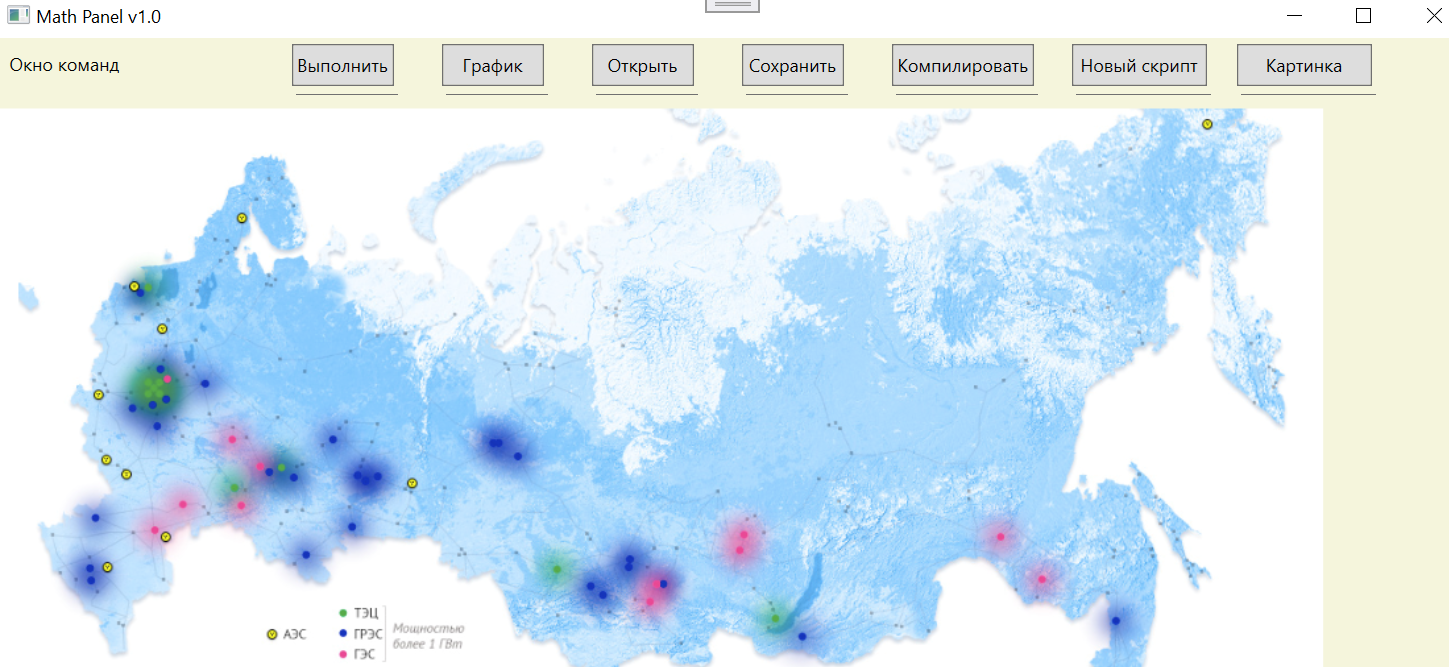


Рис.2.5. Результат работы кнопки «Картинка».

Для использования из скриптов создадим метод

//Загрузить файл с изображением в компонент

public static void SetBitmapImage(string path)

{

if (!bReady || dispObj.HasShutdownStarted) return;

//мы запускаем код в UI потоке

dispObj.Invoke(delegate

{

imgStatic.Source = new System.Windows.Media.Imaging.BitmapImage(new Uri(path, UriKind.Absolute));

});

}

Пример слайд-шоу

//test47\_slides

using MathPanel;

using System;

using System.Collections.Generic;

///сборки

///[DLL]System.dll,System.Xaml.dll,WindowsBase.dll,PresentationFramework.dll,PresentationCore.dll,System.Drawing.dll,System.Net.dll,System.Net.Http.dll,System.Core.dll[/DLL]

///

namespace DynamoCode

{

public class Script

{

public void Execute()

{

Dynamo.Console("test47\_slides");

//путь к папке

string sDir = @"C:\c\_devel\images\";

//массив файлов-изображений

string[] fnames = { "pat1\_rot.png", "pat2\_rot.png", "pat3\_rot.png", "pat4\_rot.png", "pat5\_rot.png",

"pat6\_rot.png", "pat7\_rot.png", "pat8\_rot.png", "pat9\_rot.png", "pat10\_rot.png",

"pat11\_rot.png", "pat12\_rot.png"

};

//по всем файлам

for (int i = 0; i < fnames.Length; i++)

{

var fn = fnames[i];

//загрузить файл в компонент Image

Dynamo.SetBitmapImage(sDir + fn);

//заснуть на 500 мсек

System.Threading.Thread.Sleep(500);

}

}

}

}

Показываем по очереди изображения из файлов, «засыпаем» на 500 миллисекунд.

Для надежности работы добавим в конструктор

this.Closing += OnBeforeClosing;

и обработчик

void OnBeforeClosing(object sender, EventArgs e)

{

bReady = false;

foreach (var thr in lstThr)

{

if (thr.IsAlive) thr.Abort();

}

}

Т.е. мы заканчиваем все продолжающие работу потоки, созданные при выполнении скриптов.

## 2.8. Полировка прототипа, чтение конфигурации

Целью разработки является избегать явного задания в программе значений, которые могут изменяться от пользователя к пользователю. И требовать перекомпиляции программы. Тут нам поможет конфигурационный файл app.config. Добавим в него

<appSettings>

<!--path to framework -->

<add key="netframworkpath" value="C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.7.2\" />

</appSettings>

В программу добавим пространство имен

using System.Configuration;

В конструктор добавим код чтения переменной конфигурационного файла

string s = ConfigurationManager.AppSettings["netframworkpath"];

if (!string.IsNullOrEmpty(s))

frmPath = s;

А также ссылку на сбоку System.Configuration. Теперь прототип полностью готов.