Плагины

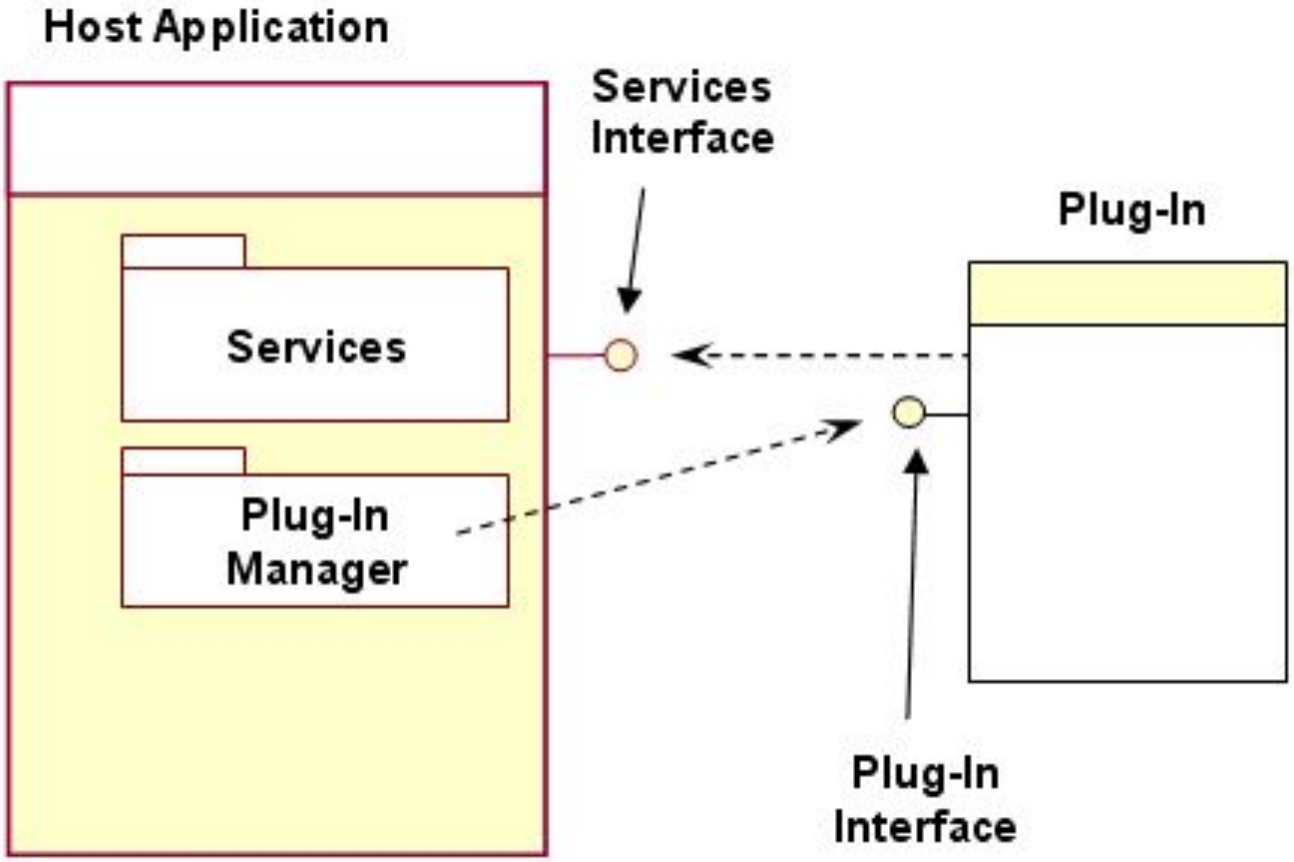
Причины создания плагинов

- Чтобы расширить функциональные возможности приложения без необходимости перекомпиляции или повторного распространения его среди заказчиков.
- Чтобы добавить функциональные возможности без необходимости доступа к оригинальному исходному коду.
- Бизнес-правила для приложения меняются часто или часто добавляются новые правила.

Этапы создания плагина

- Создание подключаемой инфраструктуры
- Создание подключаемого модуля
- Создание экземпляров подключаемых модулей и их инициализация
- Инициализация подключаемых модулей

Plugin архитектура



Применение рефлексии для создания плагинов

В .NET написание плагинов решается с помощью рефлексии (reflection).

Рефлексия позволяет динамически загружать сборки, получать информацию о методах, свойствах, событиях и полях классов из сборок, создавать новые типы и вызывать методы во время выполнения.

Среда .NET позволяет получить ссылку на описание типа по его полному имени

Классы и интерфейсы для рефлексии находятся в пространстве имен System.Reflection.

Класс Type

Основной класс для динамического получения информации о классах, интерфейсах, их полях, методах и перечислениях - Type. Для получения объекта Type можно воспользоваться несколькими разными методами:

- статический метод Type.GetType, который по имени типа возвращает объект Type
- методы GetInterface, GetInterfaces, FindInterfaces, GetElementType и GetTypeArray класса Type
- методы GetType, GetTypes и GetExportedTypes класса Assembly
- методы GetType, GetTypes и FindTypes класса Module
- оператор typeof

Получение имени сборки – свойство Assembly

Пример

```
public class MyClass
 {
  public void Hello()
  {

Console.WriteLine(“Hello!");
 }
}

// Main method
 MyClass objclass = new
MyClass();
 objclass.Start();
```

Type MyClassType = typeof(MyClass);

Type type = objclass.GetType();

Имеется возможность перебрать все типы, определённые в сборке, и найти нужный класс.

```
foreach (Type type in
Assembly.GetExecutingAssembly().
GetTypes())
```

Создание экземпляров типов

По объекту Type можно не только определять параметры типа, но и создавать его экземпляры и вызывать их методы. Для этого также существует несколько методов:

- методы CreateInstance и CreateInstanceFrom класса Activator. Это специальный класс для создания экземпляров типов и получения ссылок на удаленные объекты. Методу CreateInstance передаются объект Type или название инстанцируемого типа, массив объектов, соответствующих параметрам конструктора типа и объекты CultureInfo. Методу CreateInstanceFrom дополнительно передается имя сборки, содержащий тип. Методы, не принимающие в качестве параметра объект Type, также возвращают wrapper's ObjectHandle
- метод CreateInstance класса Assembly, создающий тип по его имени
- метод Invoke класса ContructorInfo
- метод InvokeMember класса Type

Загрузка объекта

Имя сборки (MyAsm.dll будет MyAsm).

LoadFrom напрямую загружает сборку из файла, путь к которому передается методу, заданным строкой, или на основе информации хранящейся в объекте AssemblyName (версия, криптографический ключ, информация о культуре).

Метод LoadWithPartialName загружает сборку при неполных сведениях о ней, но пользоваться им не рекомендуется из-за непредсказуемости его работы, т.к. он был разработан для бета-тестеров .NET Framework.

Можно загружать сборки и вызовом метода Load для объектов домена AppDomain.

Например, чтобы загрузить сборку в текущий домен можно воспользоваться таким кодом

```
AppDomain.CurrentDomain.Load(assemblyName);
```

Использование интерфейсов

При создании плагинов обычно используются интерфейсы, определяющие методы и свойства, которые должны реализовываться плагином. Для получения интерфейсов, которые есть у типа, используются методы GetInterface,GetInterfaces и FindInterfaces класса Type.

Метод GetInterface по имени интерфейса позвращает объект Type для этого интерфейса или null если такого интерфейса у типа нет. Метод GetInterfaces возвращает массив объектов Type с информацией об интерфейсах.

Метод FindInterfaces возвращает массив интерфейсов, выбранных с помощью фильтра - делегата, вызываемого для каждого интерфейса.

Если класс реализует несколько интерфейсов, у которых есть методы с одинаковыми названиями, то нужно использовать метод GetInterfaceMap класса Type.

Он возвращает объект InterfaceMapping для определения соотношения методов интерфейсов и методов класса, которые их реализуют.

Вызов методов

- Обычно методы вызываются с помощью метода InvokeMember класса Type. Процесс вызова метода состоит из двух этапов - привязки, при котором находится нужный метод, и непосредственно вызова. Для вызова нужно указать
- имя метода (в качестве метода может быть обычный метод, конструктор, свойство или поле)
- битовую маску из значений BindingFlags для поиска метода. В маске можно указать тип доступа метода, тип метода (поле, свойство, ...), тип данных и пр.
- объект Binder для связывания членов и аргументов
- объект, у которого вызывается метод
- массив аргументов метода
- массив объектов ParameterModifier
- объект CultureInfo

Разработка плагинов

Для демонстрации применения рефлексии при создании плагинов предлагается тестовое приложение, состоящее из 4 проектов.

- MainApp - основное приложение, к которому будут подключаться плагины. Приложение загружает из графических файлов изображения и выводит их на форме
- Interface - определяет интерфейсы IPlugin для плагинов и IMainApp для приложений, к которым будут подключаться плагины
- RandomPlugin и ReversePlugin - плагины для добавления шума к изображениям и отражения изображения по вертикали

- Добавлены еще 2 плагина для преобразования в ЧБ

Проект Interface

Проект Interface содержит только определения двух интерфейсов. Приложение, которое подключает плагины, должно реализовывать интерфейс IMainApp. Этот интерфейс объявляет единственное свойство Image, с помощью которого плагины получают изображение и возвращают его после преобразования.

```
public interface IMainApp
    {
        Bitmap Image { get; set; }
    }
```

Интерфейс для плагинов

IPlugin содержит объявления трех свойств и одного метода для преобразования изображения.

Свойства используются для получения информации о плагинах - названия, номера версии и автора. Методу передается интерфейс IMainApp.

Если бы плагины содержали бы несколько методов для преобразования изображения, то можно было поступить другим образом - создать в плагине метод для передачи в плагин интерфейса IMainApp, чтобы не передавать его каждому методу.

Плагин тогда содержал бы в себе ссылку на главное приложение.

```
public interface IPlugin
    {
        string Name { get; }
        string Version { get; }
        string Author { get; }
        void Transform(IMainApp app);
    }
```

Основное приложение

Приложение MainApp, к которому мы будем подключать плагины, это простое windows-forms приложение для отображения графический файлов. Оно реализует интерфейс IMainApp - класс формы определен как public class Form1 : System.Windows.Forms.Form, Interface.IMainApp.

На форме находится PictureBox для вывода изображения. Для реализации интерфейса IMainApp определяем свойство Image для доступа к изображению.

```
public Bitmap Image
    {
        get { return (Bitmap)pictureBox.Image; }
        set { pictureBox.Image = value; }
    }
```

Метод FindPlugins

В конструкторе формы вызывается метод FindPlugins, который находит плагины в папке с приложением и загружает их сборки. Для поиска и загрузки применяется рефлексия.

Существует и другой подход - создать для приложения конфигурационный файл, в котором прописаны пути ко всем плагинам. При этом мы не сможем устанавливать плагины путем простого копирования сборок, что не есть хорошо.

Реализация FindPlugins()

```
//Hashtable plugins = new Hashtable();
void FindPlugins()
{
  string folder = System.AppDomain.CurrentDomain.BaseDirectory;
   string[] files = Directory.GetFiles(folder, "*.dll"); // dll-файлы в
папке
   foreach (string file in files)
        try
        {Assembly assembly = Assembly.LoadFile(file);
          foreach (Type type in assembly.GetTypes())
           {
              Type iface = type.GetInterface("Interface.IPlugin");
              if (iface != null)
              {Interface.IPlugin plugin =
(Interface.IPlugin)Activator.CreateInstance(type);
          plugins.Add(plugin.Name, plugin);
             }
           }
      }
      catch (Exception ex{
   MessageBox.Show("Ошибка загрузки плагина\n" + ex.Message);
        }
}
```

Создание меню

После того, как все плагины найдены, создаем для них в функции CreatePluginsMenu пункты меню. Названия пунктов меню берутся из ключей в хеш-таблице.

Для обработки событий от меню для вызова плагинов создается обработчик OnPluginClick. В обработчике определяется названия пункта меню, который выбрал пользователь, и по нему, как по ключу в хеш-таблице, получаем интерфейс IPlugin соответствующего плагина.

У плагина вызывается метод Transform, в качестве параметра this (т.к. класс формы наследуется от интерфейса IMainApp).

CreatePluginsMenu()

```
void CreatePluginsMenu()
{
// создаем обработчик для комманд меню для плагинов
EventHandler handler = new EventHandler(OnPluginClick);
foreach (string name in plugins.Keys)
      {
      MenuItem item = new MenuItem(name, handler);
      menuItemPlugins.MenuItems.Add(item);
      }
    }
private void OnPluginClick(object sender, EventArgs args)
{
Interface.IPlugin plugin
     =(Interface.IPlugin)plugins[((MenuItem)sender).Text];
      plugin.Transform(this);
}
```

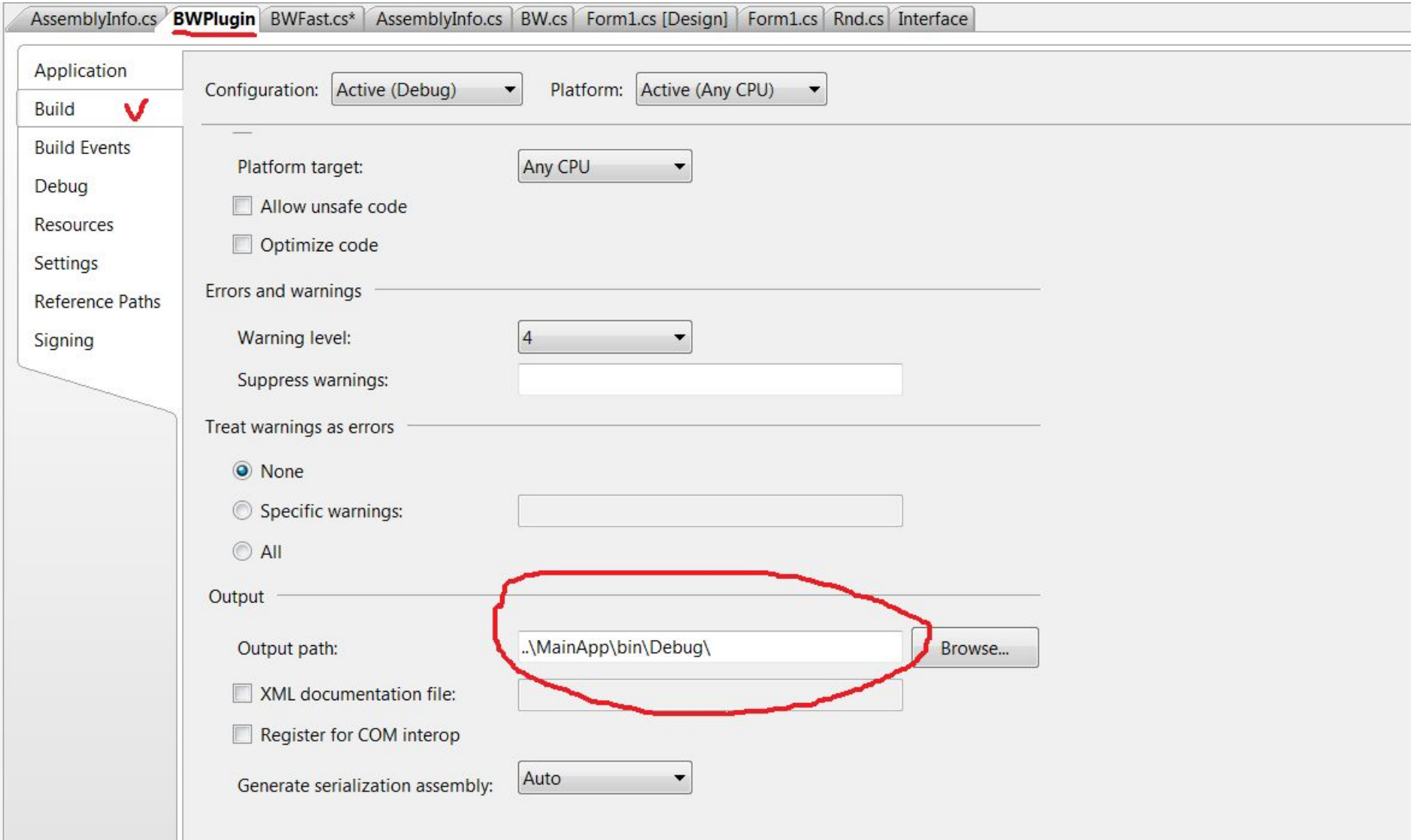
Создание плагина

Для создания плагинов создаем новый проект ClassLibrary, добавляем ссылку на сборку Interface и реализуем свойства и методы интерфейса IPlugin.

В примере разработаны 2 плагина: ReverseTransform для отражения изображения по вертикали;

RandomTransform для внесения случайного шума в изображение.

Properties


AssemblyInfo.cs
BWPlugin
BWFast.cs*
AssemblyInfo.cs
BW.cs
Form1.cs [Design]
Form1.cs
Rnd.cs
Interface
Application
Build
Build Events
Debug
Resources
Settings
Reference Paths
Signing
Configuration: Active (Debug)
Platform: Active (Any CPU)
Platform target: Any CPU
Allow unsafe code
Optimize code
Errors and warnings
Warning level: 4
Suppress warnings:
Treat warnings as errors
None
Specific warnings:
All
Output
Output path: ..\MainApp\bin\Debug\
Browse...
XML documentation file:
Register for COM interop
Generate serialization assembly: Auto

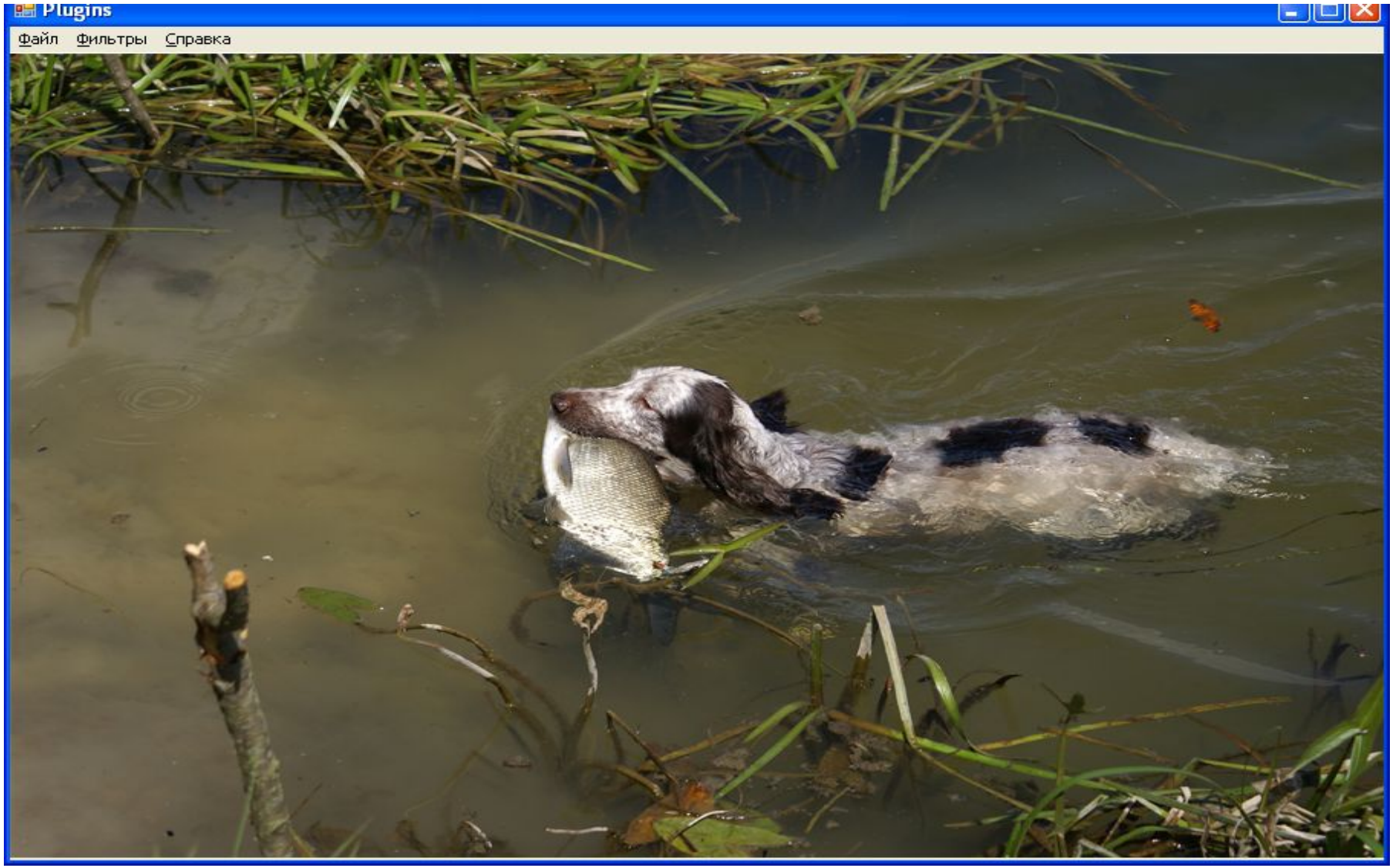
ReverseTransform

```
public class ReverseTransform : Interface.IPlugin
    {
        public string Name
        { get { return "Переворот изображения"; }}

        public string Version
        {get { return "1.0"; } }

        public string Author
        {get { return «XXX"; } }

       public void Transform(Interface.IMainApp app)
        {
            Bitmap bitmap = app.Image;
            for (int i = 0; i < bitmap.Width; ++i)
                for (int j = 0; j < bitmap.Height / 2; ++j)
                {
                 Color color = bitmap.GetPixel(i, j);
                 bitmap.SetPixel(i, j, bitmap.GetPixel(i, bitmap.Height - j - 1));
                 bitmap.SetPixel(i, bitmap.Height - j - 1, color);
         }
         app.Image = bitmap;
        }
    }
```

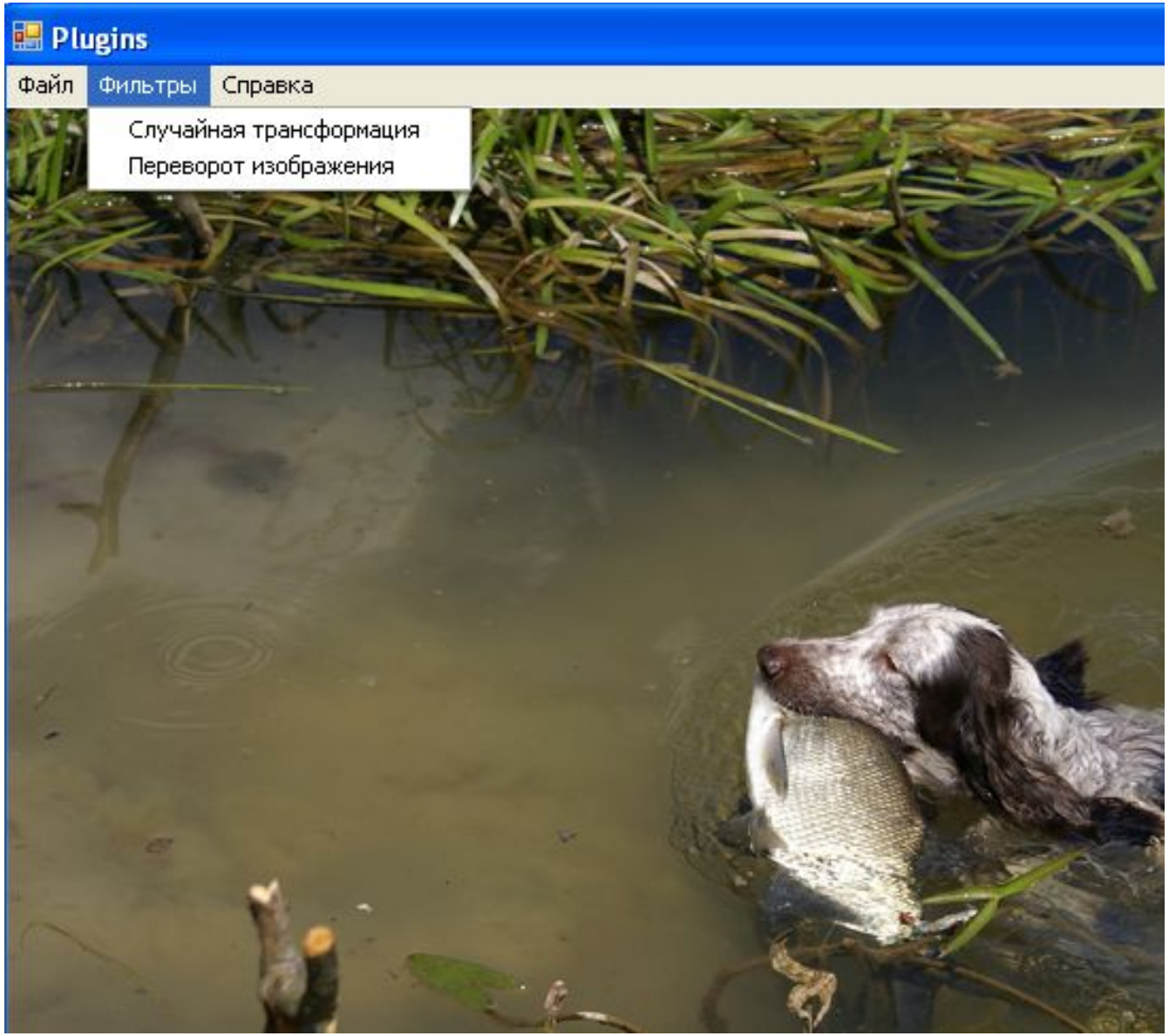
RandomTransform

```
/// Плагин для вставки случайных точек в рисунок.
    public class RandomTransform : Interface.IPlugin
    {
    public string Name{ get { return "Случайная трансформация"; } }

    public string Version{get { return "1.0"; }}

    public string Author {get { return «YYY"; }}

    public void Transform(Interface.IMainApp app)
     {
    Bitmap bitmap = app.Image;
    Random rand = new Random(DateTime.Now.Millisecond);
    int pixels = (int)(0.1 * bitmap.Width * bitmap.Height);
    for (int i = 0; i < pixels; ++i)
      bitmap.SetPixel(rand.Next(bitmap.Width - 1), rand.Next(bitmap.Height),
          Color.FromArgb(rand.Next(255), rand.Next(255), rand.Next(255)));
    app.Image = bitmap;
      }
    }
```

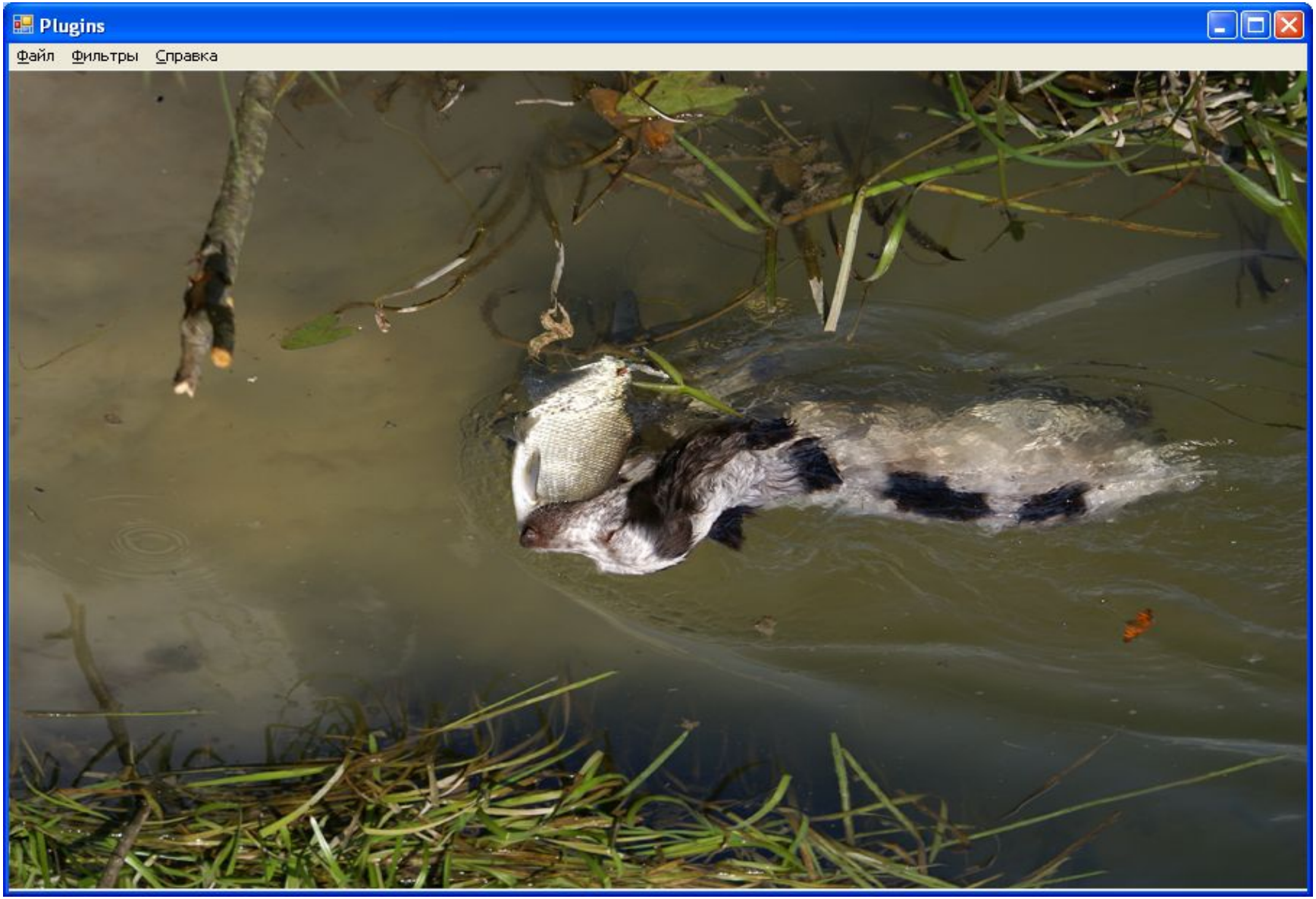
Загрузка изображения



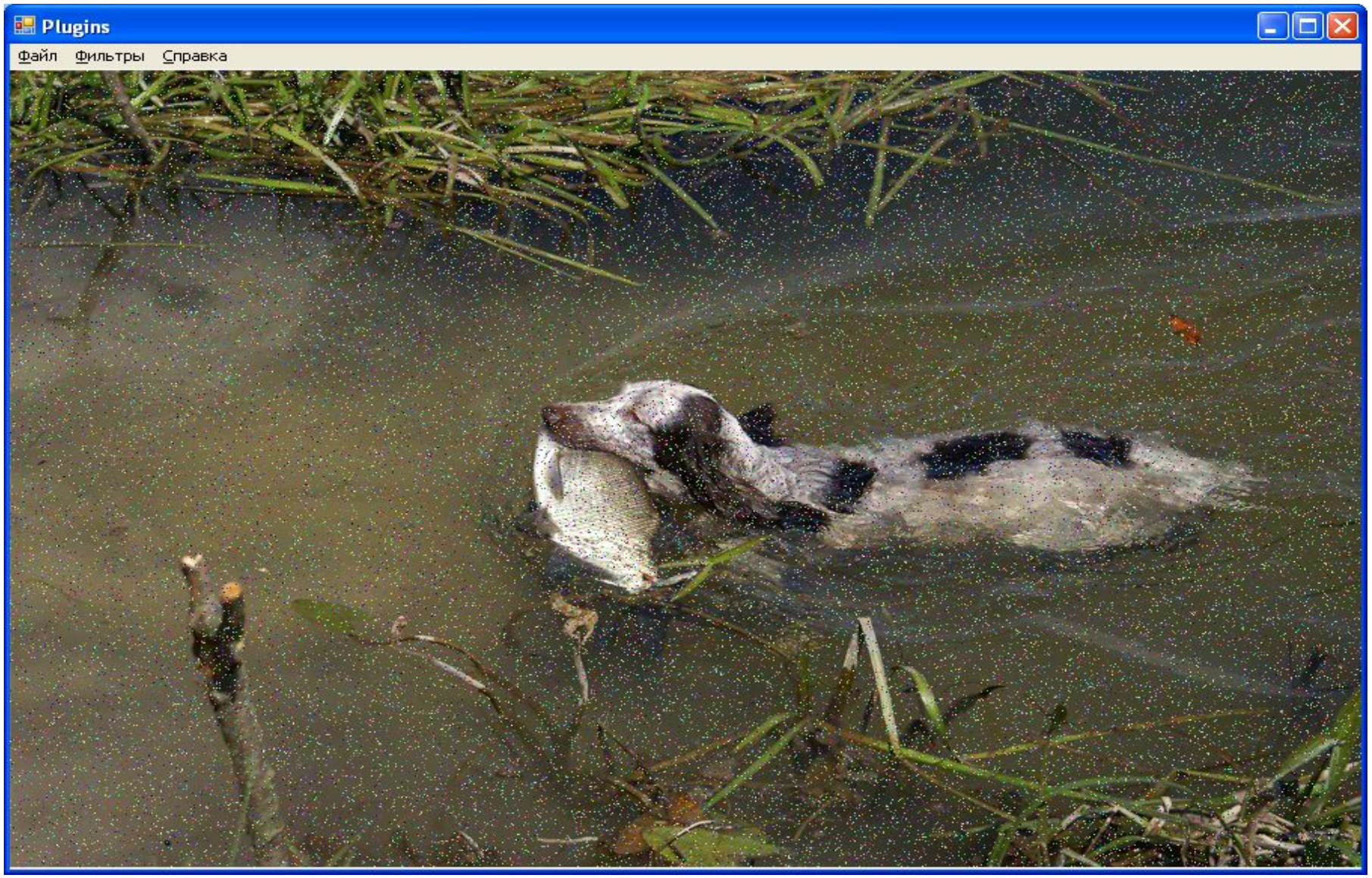
Меню


Plugins
Файл
Фильтры
Справка
Случайная трансформация
Переворот изображения

Поворот изображения



Случайная трансформация



Ускорение работы с изображениями

- Локирование данных
- Вычисление размера массива
- Копирование данных в массив
- Работа с данными
- Возвращение данных в Bitmap
- Разлокирование данных

Локирование данных

```
Rectangle rect = new Rectangle(0, 0, bmp.Width,
    bmp.Height);

System.Drawing.Imaging. BitmapData bmpData =
    bmp.LockBits(rect,System.Drawing.Imaging.ImageLock
    Mode.ReadWrite, bmp.PixelFormat);
///////
```

Вычисление размера массива

```
// Get the address of the first line.
            IntPtr ptr = bmpData.Scan0;

// Declare an array to hold the bytes of the bitmap.
 int bytes  = Math.Abs(bmpData.Stride) *
    bmp.Height;
byte[] rgbValues = new byte[bytes];
```

Шаг по индексу — ширина ряда пикселей (строки развертки), округленного до четырех байтов.

Копирование данных в массив

```
// Copy the RGB values into the array.
System.Runtime.InteropServices.Marshal.Copy(ptr,
  rgbValues, 0, bytes);
```

===

Предоставляет коллекцию методов для выделения неуправляемой памяти, копирования блоков неуправляемой памяти и преобразования управляемых типов в неуправляемые, а также прочих разнообразных методов, используемых при взаимодействии с неуправляемым кодом.

Возвращение данных в Bitmap

```
System.Runtime.InteropServices.Marshal.Copy(rgbValues, 0, ptr, bytes);
```

Разлокирование данных

```
bmp.UnlockBits(bmpData);
```