ПРОГРАММНЫЙ КОМПЛЕКС «АТОМ»

ИНСТРУКЦИЯ РАЗРАБОТЧИКА

2023

Терминология

В данном руководстве используются следующие термины:

АТОМ – Автоматическое тестирование объектов мониторинга

ОС - Операционная система

КСПД - Комплексная среда передачи данных

VPN - Virtual Private Network

АРМ - Автоматизированное рабочее место

Термин	Пояснение
АТОМ	Автоматическое тестирование объектов мониторинга
ОС	Операционная система
КСПД	Комплексная среда передачи данных
VPN	Virtual Private Network
DB	Data Base
Пользователи Системы	
АРМ	Автоматизированное рабочее место
Площадка ФТ	Сервер функционального тестирования, предназначенный для запуска тестов в режиме имитации действий пользователя

Общие действия по разработке сценариев

1. Необходимое программное обеспечение для АРМ

Разработку сценариев необходимо вести на языке программирования C#.

В качестве среды разработки рекомендуется использовать Visual Studio 2017 или новее.

2. Описание необходимых библиотек

Для написания сценариев необходимо пользоваться следующими библиотеками:

Atom.Core.dll – библиотека содержит основной набор классов, необходимый для работы системы.

Atom.Script.dll – библиотека предоставляет управление сценарием.

Atom.WebApp - библиотека содержит набор классов для работы с Web-приложениями. Данная библиотека является надстройкой над Selenium (который в большинстве случаев используется для тестирования Web-приложений). Для работы Selenium необходим следующий библиотеки: SeleniumExtras.WaitHelpers.dll, WebDriver.dll, WebDriver.Support.dll. Под каждый браузер также требуется отдельный Web-драйвер. Например, для работы с сайтом через браузер Google Chrome, требуется файл chromedriver.exe.

Atom.BD.PostgreSQL.dll – библиотека содержит набор классов для работы с БД PostgreSQL, т.к. эта БД выступает в качестве основного хранилища данных. Также, потребуется библиотека Npgsql.dll.

Atom.BD.SQLite.dll – библиотека содержит набор классов для работы с БД SQLite, т.к. эта БД выступает в качестве локального хранилища данных (когда отсутствует доступ до основной БД PostgreSQL). Также, потребуется библиотека System.Data.SQLite.dll.

Необходимые файлы находятся в каталоге _lib.

3. Использование решения Template

Для упрощения создания сценария, подготовлено решение «Template». Он состоит из двух проектов: template_dll и template_exe.

Решение template_dll содержит основу для написания сценария. Оно состоит из трех файлов: PageElement.cs, PageAction.cs и Script.cs.

Файл PageElement.cs содержит класс PageElement, который должен содержать список элементов, над которыми будут выполняться действия.

Содержимое файла PageElement.cs:

```
using System;

namespace PageObject
{

   //** Класс содержит список элементов, над которыми выполняются действия.
   public class PageElement
   {

   }
}
```

Файл PageAction.cs содержит класс PageAction, который должен содержать список действий над элементами, описанных в классе PageElement.

Содержимое файла PageAction.cs:

```
using System;
using Atom.Core;

namespace PageObject
{
   //** Класс содержит список действий (шагов) над элементами
   public class PageAction
   {
      PageElement pageElement = new PageElement();
      ParameterCollection parameters;

      // Инициализация параметров
      public void InitParam(ParameterCollection parameters)
```

```
        {
            this.parameters = parameters;
        }
    }
}
```

Файл Script.cs содержит класс Script, который содержит порядок выполнения действий, описанных в классе PageAction, над элементами, описанных в классе PageElement.

Содержимое файла Script.cs:

```
using System;
using Atom.ScriptManager;


namespace PageObject
{
    //** Класс содержит порядок выполнения шагов (набор действий над элементами).
    //      Порядок выполнения шагов следует задать в методе "Executable()".
    //       Порядок выполнения шагов, которые необходимо обязательно выполнить после основного скрипта, следует задать в методе "ExecutableFinally()".
    public class ScriptMonitoring : Script
    {
        PageAction pageAction = new PageAction();

        //!!! Необходим пустой конструктор ScriptMonitoring для dll

        protected override void Executable()
        {
            //!!! Параметры в dll попадают только при вызове Execute, а не при вызове конструктора класса PageObject.ScriptMonitoring
            pageAction.InitParam(Parameters);
        }

        protected override void ExecutableFinally()
        {

        }
    }
}
```

Решение template_exe содержит основу для вызова и демонстрации работы сценария.

Содержимое файла Form1.cs:

```
using System;
using System.Windows.Forms;
using PageObject;

namespace template_exe
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      ScriptMonitoring script;

      private void btnStart_Click(object sender, EventArgs e)
      {
         script = new ScriptMonitoring();

         //** Название сценария
         //     Также, данное имя задает название каталога для логирования ошибок и отладочной информации.
         script.Name = "Template";

         //** Ссылка на форму.
         //     Следует использовать при асинхронной работе с объектом Script, когда в обработчиках событий Script задействованы компоненты формы.
         script.MainForm = this;

         //** Задаем ограничение на время выполнения сценария.
         script.SetTimeout(60);

         //** Разрешаем сохранять в файл результат работы сценария.
         script.EnabledSavingResultToFile = true;

         //**Запрещаем сохранять в БД результат работы сценария.
         //     Если разрешить, то к проекту следует подключить следующие библиотеки: Atom.BD.PostgreSQL.dll, Npgsql.dll.
         //     Если к основной БД доступа не будет, то будет произведена попытка сохранить данные в локальную БД (SQLite),
         //     для этого потребуются следующие библиотеки: Atom.BD.SQLite.dll, System.Data.SQLite.dll.
         script.EnabledSavingResultToBD = false;

         //** Параметры, необходимые для логирование результатов работы сценария в БД (режим script.EnableSavingResultToBD = true)
         //script.Parameters.Add(Config.ParamNames.Aut.ID, 1);
```

```
            //script.Parameters.Add(Config.ParamNames.Aut.CodeID, 1);
            //script.Parameters.Add(Config.ParamNames.Server.ID, 1);

            //** Параметры, необходимые для работы сценария
            script.Parameters.Add("ParamName", null);

            //** Событие возникает до начала работы скрипта.
            script.BeforeExecute += (scr, ev) =>
            {
                btnStart.Enabled = false;
                btnStop.Enabled = true;
                txtLog.Text += $"---------- Начало выполнения сценария {scr.Name} ----------
\r\n";
            };

            //** Событие возникает до начала выполнения шага.
            script.BeforeExecuteStep += (s, ev) =>
            {
                txtLog.Text += s.Name + "\r\n";
            };

            //** Событие возникает после завершения выполнения шага.
            script.AfterExecuteStep += (s, ev) =>
            {
                txtLog.Text += s.ToStringDetail() + "\r\n" + "\r\n";
            };

            //** Событие возникает после завершения работы скрипта.
            script.AfterExecute += (scr, ev) =>
            {
                btnStart.Enabled = true;
                btnStop.Enabled = false;
                txtLog.Text += $"---------- Конец работы сценария ----------\r\n";
            };

            //** Запускаем сценарий.
            script.ExecuteAsync();
        }

        private void btnStop_Click(object sender, EventArgs e)
        {
            //** Принудительная остановка сценария.
            script?.Stop();
        }
    }
}
```

Необходимые файлы находятся в каталоге \Template.

4. Использование решения Example

В качестве примера использования шаблона Template, приведем код для тестирования сайта «https://www.rt.ru».

Содержимое файла PageElement.cs:

```
using System;
using Atom.WebApp.WebElementManager;

namespace PageObject
{
   //** Класс содержит список элементов, над которыми выполняются действия.
   public class PageElement
   {
      //** divLogo - признак загрузки гл.страницы
      public WebElement divLogo = new WebElement(By.ClassName("header__logo"));
   }
}
```

Содержимое файла PageAction.cs:

```
using System;
using Atom.Core;
using Atom.Core.ParametersHelperForAut;
using Atom.ScriptManager;
using Atom.WebApp.WebBrowserManager;

namespace PageObject
{
   //** Класс содержит список действий (шагов) над элементами
   public class PageAction
   {
      PageElement pageElement = new PageElement();
      ParameterCollection parameters;

      // Инициализация параметров
      public void InitParam(ParameterCollection parameters)
      {
         this.parameters = parameters;
      }

      public StepResult Start()
      {
```

```
            string nameStep = "Запуск браузера";
            StepResult step = new StepResult(nameStep);
            try
            {
                step.Result = WebBrowser.Start(parameters.GetAutWebBrowser(), null, null,
(int)parameters["PageLoadTimeout", 60]);

                if (step.Result.IsOK)
                    WebBrowser.Window.Maximize();
            }
            catch (Exception ex)
            {
                step.Result.Exception = ex;
                step.Result.ErrorText = $"Критическая ошибка при выполнении шага
'{nameStep}'";
            }
            return step;
        }

        public StepResult OpenUrl()
        {
            string nameStep = "Открытие сайта";
            StepResult step = new StepResult(nameStep);
            try
            {
                step.Action = "Открываем URL: " + parameters.GetAutUrl();
                WebBrowser.Url = parameters.GetAutUrl();

                step.Action = "щем контрольной элемент (логотип) на гл.странице";
                step.Result = pageElement.divLogo.Exists();
            }
            catch (Exception ex)
            {
                step.Result.Exception = ex;
                step.Result.ErrorText = $"Критическая ошибка при выполнения шага
'{nameStep}'";
            }
            return step;
        }

        public void End()
        {
            try
            {
                WebBrowser.Quit();
            }
            catch
            {
```

```
        }
    }
}
}
```

Содержимое файла Script.cs:

```
using System;
using Atom.ScriptManager;

namespace PageObject
{
    //** Класс содержит порядок выполнения шагов (набор действий над элементами).
    //      Порядок выполнения шагов следует задать в методе "Executable()".
    //      Порядок выполнения шагов, которые необходимо обязательно выполнить после основного скрипта, следует задать в методе "ExecutableFinally()".
    public class ScriptMonitoring : Script
    {
        PageAction pageAction = new PageAction();

        //!!! Необходим пустой конструктор ScriptMonitoring для dll

        protected override void Executable()
        {
            //!!! Параметры в dll попадают только при вызове Execute, а не при вызове конструктора класса PageObject.ScriptMonitoring
            pageAction.InitParam(Parameters);

            ExecuteStep("Запуск браузера", pageAction.Start);
            ExecuteStep("Открытие сайта", pageAction.OpenUrl);
        }

        protected override void ExecutableFinally()
        {
            ExecuteStepFinally(pageAction.End);
        }
    }
}
```

Содержимое файла Form1.cs:

```
using System;
using System.Windows.Forms;
using Atom.Core;
using PageObject;

namespace basic_template
{
```

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    ScriptMonitoring script;

    private void btnStart_Click(object sender, EventArgs e)
    {
        script = new ScriptMonitoring();

        Config.Highlighting.WebApp.Highlight = chHighlight.Checked;

        //** Название сценария
        //      Также, данное имя задает название каталога для логирования ошибок и отладочной информации.
        script.Name = "Example";

        //** Ссылка на форму.
        //      Следует использовать при асинхронной работе с объектом Script, когда в обработчиках событий Script задействованы компоненты формы.
        script.MainForm = this;

        //** Задаем ограничение на время выполнения сценария.
        script.SetTimeout(60);

        //** Разрешаем сохранять в файл результат работы сценария.
        script.EnabledSavingResultToFile = true;

        //**Запрещаем сохранять в БД результат работы сценария.
        //      Если разрешить, то к проекту следует подключить следующие библиотеки: Atom.BD.PostgreSQL.dll, Npgsql.dll.
        //      Если к основной БД доступа не будет, то будет произведена попытка сохранить данные в локальную БД (SQLite),
        //      для этого потребуются следующие библиотеки: Atom.BD.SQLite.dll, System.Data.SQLite.dll.
        script.EnabledSavingResultToBD = false;

        //** Параметры, необходимые для логирование результатов работы сценария в БД (режим script.EnableSavingResultToBD = true)
        //script.Parameters.Add(Config.ParamNames.Aut.ID, 1);
        //script.Parameters.Add(Config.ParamNames.Aut.CodeID, 1);
        //script.Parameters.Add(Config.ParamNames.Server.ID, 1);

        //** Параметры, необходимые для работы сценария
        script.Parameters.Add("WebBrowser", "GoogleChrome");
        script.Parameters.Add("Url", @"https://www.rt.ru");
```

```
            script.Parameters.Add("PageLoadTimeout", 120);

            //** Событие возникает до начала работы скрипта.
            script.BeforeExecute += (scr, ev) =>
            {
                btnStart.Enabled = false;
                btnStop.Enabled = true;
                txtLog.Text += $"---------- Начало выполнения сценария {scr.Name} ----------
\r\n";
            };

            //** Событие возникает до начала выполнения шага.
            script.BeforeExecuteStep += (s, ev) =>
            {
                txtLog.Text += s.Name + "\r\n";
            };

            //** Событие возникает после завершения выполнения шага.
            script.AfterExecuteStep += (s, ev) =>
            {
                txtLog.Text += s.ToStringDetail() + "\r\n" + "\r\n";
            };

            //** Событие возникает после завершения работы скрипта.
            script.AfterExecute += (scr, ev) =>
            {
                btnStart.Enabled = true;
                btnStop.Enabled = false;
                txtLog.Text += $"---------- Конец работы сценария ----------\r\n\r\n";
            };

            //** Запускаем сценарий.
            script.ExecuteAsync();
        }

        private void btnStop_Click(object sender, EventArgs e)
        {
            //** Принудительная остановка сценария.
            script?.Stop();
        }

        private void btnClear_Click(object sender, EventArgs e)
        {
            txtLog.Clear();
        }
    }
}
```

Необходимые файлы находятся в каталоге \Example.

Файлы к
руководству разрабс