AssistEdge RPA Community Edition 17.5.5 Customization Guide

Public

COPYRIGHT NOTICE

DISTRIBUTION LIST

(This document is to be accessed and distributed only for the below recipients)

Organization	Team/ Recipient details
EdgeVerve Systems Limited	Users of the AssistEdge RPA Community Edition

Contents

1 Introduction

The AssistEdge RPA Community Edition is a lean version of the AssistEdge RPA product with powerful automation capabilities and easy-to-use user interface. The feature-rich AssistEdge Automation Studio is a key component of the Community Edition. It enables users to create and configure process automations. The Automation Studio includes automation configuration capabilities for varied and heterogeneous technology applications.

1.1 About this Guide

This Customization Guide contains steps that enables user to configure various automations which are not configurable using the Automation Studio. It mainly focuses on configuration related to 17.x release.

2 Configuration in Automation

This section provides steps that enable the user to configure various automations that are not configurable through Automation Studio for the 17.x release.

The automations are:

- Custom Application
- Code Editor
- Micro Bot

2.1 Creating DLL for Custom Application

1. In Visual Studio, create new project with output type as **Class Library** using .NET framework 4.6.

> **Note:** Make sure that DLL is built using x86 platform and **DLL name is same as Namespace**.

2. Add *Proton.Utilities.CodePluginInterfaces*, *PROTON.Utilities.Interfaces*, and *Proton.Miscellaneous.Framework* DLLs (present in EVA bot directory) references in the project.
3. Add reference to ***PresentationFramework*** and ***WindowsBase*** to use the Dispatcher to access the main thread when required.
4. Implementing ***ICodeplugin*** interface gets following methods:
    a. Launch – is used to launch custom applications.
    b. DoSignIn – is used to perform sign in for custom applications.
    c. DoAutomation – is used to perform automation in custom applications.
    d. DoReset – is used to perform reset automation in custom applications.
    e. DoDispose – is used to dispose custom applications.
5. Once the launch is complete:
    a. Raise the *LaunchCompleted* event.
    b. Pass the code plugin object as sender.
    c. Set the application object in the *appContext InstanceObject*, set *IsLaunched* as True and pass the *appContext* object in the *LaunchEventArgs* in the *LaunchCompleted* event.

> **Note:** Raise all the events in main thread, i.e. using Dispatcher.

Custom App *Launch*:

```
public ApplicationContextForAutomation Launch()
        {
            //Code to launch the custom application goes here

            //Events for handling form size change in SE
            this.SizeChanged -= MyCustomApp_SizeChanged;
            this.SizeChanged += MyCustomApp_SizeChanged;

            //Set application context
            applicationContextFormAutomation = new ApplicationContextForAutomation();
            applicationContextFormAutomation.InstanceName = "MyCustomApp";
            //Assign your application instance object in the context
            applicationContextFormAutomation.InstanceObject = objIE;
            applicationContextFormAutomation.InstanceType =
objIE.GetType().ToString();

            LaunchEventArgs launchEventArgs = new LaunchEventArgs();
            launchEventArgs.appContext = applicationContextFormAutomation;

            //Code to dock the application in SE
            System.Windows.Application.Current.Dispatcher.BeginInvoke((Action)delegate
()
            {
                if (SEFramework.UseDefaultDockBehaviour)
                {
                    AddNewApplicationTabUIWithObject("MyCustomApp", this.Handle,
this);
                    Proton.Miscellaneous.Framework.Win32.SetParent(childHandle,
this.Handle);

Proton.Miscellaneous.Framework.Win32.ShowWindow((IntPtr)childHandle, 3);
                    this.Show();
                }
                this.LaunchCompleted(this, launchEventArgs);
            });

            return null;
        }
```

Custom App Launch

6. In ***SizeChanged*** event generated as shown below, set height and width of the application to be equal to that of the *usercontrol*.

Custom App *SizeChanged*:

```
private void MyCustomApp_SizeChanged(object sender, EventArgs e)
        {
            if (objIE != null)
            {
                objIE.Height = this.Height;
                objIE.Width = this.Width;
            }
               }
```

Custom App SizeChanged

7. Once *LaunchCompleted* event is raised:

    a. EVA bot calls the *DoSignIn* method of the code plugin if *SignIn* is configured for the Custom Application.

    b. Raise the *SignInCompleted* event once the sign in is done with *SignInEventArgs* having either *SignInStatus* as Completed or Failed using *SignInStatus* Enumeration.

If this method takes more time than the value configured in the sign-in manager in IDE for the code plugin, then EVA bot marks the sign-in as failed for this custom application. User gets User Id and password in *inputParamDict.*

Custom App *DoSignIn*:

```
public bool DoSignIn(Dictionary<string, string> inputParamDict, ref Dictionary<string,
string> outPutParamDict, ApplicationContextForAutomation appContext)
        {
            //Code to perform sign in into your application goes here


            //On successful sign in, raise the SignInCompleted event
            SignInEventArgs signInEventArgs = new SignInEventArgs();
            signInEventArgs.SignInStatus = SignInStatus.SignInCompleted;
            System.Windows.Application.Current.Dispatcher.BeginInvoke((Action)delegate
()
            {
                SignInCompleted(this, signInEventArgs);
            });

            return true;
               }
```

Custom App DoSignIn

8. Application automation logic is to be implemented in *DoAutomation* method.

- *InputParamDict* contains all inputs parameters configured for this custom application in format of *KeyValuePair,* where keys correspond to field name and value corresponds to its value.
- *OutPutParamDict* contains *KeyValuePair* of all the output parameters, where keys correspond to field name and value corresponds to its value which is required to be modified or assigned by user depending on the output from custom application.
- To return the extracted fields to EVA bot: Raise *FillExtractedFieldsEvent* with *ExtactedFieldsEventArgs* having dictionary containing *extractedFields* assigned equal to *outputPutParamDict*.

Note: Ensure this event is raised using Dispatcher.

9. To mark search completed or failed, raise *SearchStatusEvent* with *SearchStatusEventArgs* having either *searchCompleted* or *SearchFailed* as enumeration in *eventArgs*.

Custom App *DoAutomation*:

```
public Dictionary<string, string> DoAutomation(Dictionary<string, string>
inputParamDict, ref Dictionary<string, string> outputParamDict,
ApplicationContextForAutomation appContext)
        {
            //Code to automate the application goes here; inputParamDict will have the
input data configured for automation


            //Raise event for filling extracted fields
            ExtactedFieldsEventArgs extractedFieldEventArgs = new
ExtactedFieldsEventArgs();
            extractedFieldEventArgs.ExtractedFields = new Dictionary<string, string>();
            extractedFieldEventArgs.ExtractedFields.Add("Key from outputParamDict",
"Extracted value for the key");
            System.Windows.Application.Current.Dispatcher.BeginInvoke((Action)delegate
()
            {
                FillExtractedFieldsEvent(this, extractedFieldEventArgs);
            });


            //Raise SearchStatusEvent when search completed with appropriate status
            SearchStatusEventArgs searchStatusEventArgs = new SearchStatusEventArgs();
            searchStatusEventArgs.SearchStatus = SearchStatus.SearchCompleted;
            System.Windows.Application.Current.Dispatcher.BeginInvoke((Action)delegate
()
            {
                SearchStatusEvent(this, searchStatusEventArgs);
            });


            return outputParamDict;
               }
```

Custom App DoAutomation

10. Custom Application is reset whenever the user resets the process in EVA bot, to achieve that need to implement *DoReset* method. After reset, user has to raise *SearchStatusEvent* as *ResetCompleted* or *ResetFailed* using *SearchStatus* Enumeration.

Note: Ensure that this event is to be raised using Dispatcher as shown below.

Custom App *DoReset*:

```
public void DoReset(string applicationName, Dictionary<string, object> inputParamDict)
        {
            //Code to reset the application goes here. This method gets called on Reset
button click in SE


            //Raise SearchStatusEvent on successful reset
            SearchStatusEventArgs searchStatusEventArgs = new SearchStatusEventArgs();
            searchStatusEventArgs.SearchStatus = SearchStatus.ResetCompleted;
            System.Windows.Application.Current.Dispatcher.BeginInvoke((Action)delegate
()
            {
                SearchStatusEvent(this, searchStatusEventArgs);
            });
              }
```

Custom App DoReset

11. Custom Application has to be reloaded whenever user reloads the applications in EVA bot, to achieve that user needs to implement *DoDispose* method. Whenever Reload is clicked, *DoDispose* method is called first and then launched.

Note: *DoDispose* method is used to dispose the customization and reset the application.

Custom App *DoDispose*:

```
public void DoDispose(string applicationName, Dictionary<string, object>
inputParamDict)
        {
            //Code to close and cleanup the application goes here
               }
```

Custom App DoDispose

Note: All custom plugin DLLs must be built using the supported .NET 4.6 framework.

2.2 Code Editor

The Code Editor helps the user to write custom code for a particular step in the automation. If the user is unable to automate a particular step using Automation Studio, add the Code Editor for that particular step and write custom code.

Steps to create Code Editor DLL:

1. In Visual Studio, create a new project with output type as Class Library using .NET framework 4.6.

2. Add *SE.Core.Automation.Interfaces* DLL and *SE.Core.Automation*. Models DLL references in the project.
3. Implement ICodeEditor using SE.Core.Automation.Interfaces.Common.
4. Once *ICodeEditor* is implemented, the user gets *PerformAction* method and execution completed event.
5. In *PerformAction*, the user gets *IApplicationAutomation* which contains an instance of the plugin on which the user can perform required automation.
6. In extracted fields, the user gets a dictionary of input and output arguments that are in scope of a flow chart. The data is extracted as well as set back to dictionary for further processing.
7. Once Code Editor operations are completed, the user should raise the execution completed event.

Note: To typecast *IApplicationAutomation* instance into specific application plugin, add reference of Plugin from **$RootFolder**\PA\Plugins**ApplicationType***.dll

In case of **Web Applications** for **firefox and chrome**: ApplicationType is **Selenium**.

and for **Internet explorer**: ApplicationType is **WatIn**

All custom plugin DLLs must be built using the supported .NET 4.6 framework.

8. To access the Internet Explorer instance in Code Editor from **Web IE plugin**, follow these steps:
    a. Typecast IAppicationAutomation into IEPlugin.
    b. Call method *InternetExplorerObject* of *IEPlugin* which returns instance of internet explorer.

Refer code snippet mentioned in Figure 8 for reference:

Code Editor:

```
public class MyCodeEditor : ICodeEditor
    {
        public event EventHandler ExecutionCompleted;


        public void PerformAction(IApplicationAutomation plugin, Dictionary<string,
object> extractedFields)
        {
            //Code to automate goes here
            Thread t = new Thread(() =>
            {
                try
```

```
                {
                    //access the relevant object from actual application type plugin by
casting the plugin input parameter
                    EV.AE.IEWebAutomation.IEPlugin iePlugin = plugin as
EV.AE.IEWebAutomation.IEPlugin;
                    InternetExplorer ie = iePlugin.InternetExplorerObject();
                }
                catch(Exception ex)
                { }
            });
            t.SetApartmentState(ApartmentState.STA);
            t.Start();
            t.Join();

            //Raise ExecutionCompleted event on completion of action
            ExecutionCompleted.Invoke(this, null);
        }
           }
```

Code Editor

2.3 Micro Bot

Micro bot enables the user to write code for reusable components. This is a feature to extend the product boundary by creating reusable components which are inserted anywhere in the process to perform custom logic.

Steps to create Micro-bot:

1. In Visual Studio, create a new project with output type as **Class Library** using .NET framework 4.6.
2. Add *SE.Core.Automation.Interfaces* DLL and *SE.Core.Automation.Models* DLL references in the project.
3. Implement *IAECodePackage* using *SE.Core.Automation.Interfaces.*
4. Once *IAECodePackage* is implemented, the user gets execute method and execution completed event.

Microbot:

```
public class MyMicrobot : IAECodePackage
    {
        public event EventHandler ExecutionCompleted;

        public void Execute(StudioContext context)
        {
```

```
            //Microbot code goes here

            //Raise the ExecutionCompleted event when done
            ExecutionCompleted(this, null);
        }
           }
```

Microbot

5. To map input and output parameters, custom attributes are defined to be set on .NET class properties to set or get values from output and input of micro-bot.
6. Once execution is complete, the user must raise the execution complete event.

Microbot Input Outputs:

```
[ArgumentDirection(Direction = DirectionType.Input)]
        public int Input1
        {
            get;
            set;
        }

        [ArgumentDirection(Direction = DirectionType.Output)]
        public int Output1
        {
            get;
            set;
               }
```

Microbot Input Outputs

Note: Ensure each microbot has a distinct main dll.

