This document describes the required actions on the side of the GOPlugin developer in order to start developing new plugins. The instructions are split into two sections, **PluginExampleWin8 (Windows 8)**/**PluginExampleWP8 (Windows Phone 8)** which is a subproject contained within GOPluginProject solution and the **GOPluginProjectWin8 (Windows 8)/ GOPluginProjectWP8 (Windows Phone 8)** itself.

PluginExampleWin8 (Windows 8)/PluginExampleWP8 (Windows Phone 8)

The PluginExampleWin8 (Windows 8)/PluginExampleWP8 (Windows Phone 8) C# project is serving as a template for developing GOPlugins.

The project should contain all the necessary source files and other resources that implement the desired plugin behavior. It supports **ONLY** one plugin and all source files and resources will then be deployed using the GO!AppZone build service.

For every plugin implementation there are two mandatory classes that should be implemented along with any other that the developer may wish add.

These are: **GOPlugin** and **GOPluginAction**

**GOPlugin**

A class (e.g. ‘ContactsPlugin’) that inherits from GOPlugin and defines any operations that need to be performed during the initialization process of the plugin, as well as when the object is disposed. It can also contain any properties that the developer may find useful for the execution of the plugin action and will be available throughout the object’s lifecycle.

Any operations that should be performed when the object is initializing should go into constructor:

**Windows Phone 8**

public ContactsPlugin(Microsoft.Phone.Controls.PhoneApplicationFrame frame)

: base(frame)

{

}

**Windows 8**

public ContactsPlugin(MyToolkit.Paging.Frame frame)

: base(frame)

{

}

Any operations that should be performed when the application is launching should go into method:

**Windows8\Windows Phone 8**

public override void Application\_Launching()

{

}

Any operations that should be performed when the application is activated should go into method:

**Windows8\Windows Phone 8**

public override void Application\_Activated ()

{

}

Any operations that should be performed when the application is deactivated should go into method:

**Windows8\Windows Phone 8**

public override void Application\_Deactivated()

{

}

Any operations that should be performed when the application is closing should go into method:

**Windows8\Windows Phone 8**

public override void Application\_Closing()

{

}

**GOPluginAction**

A class (e.g. ‘ContactsAction) that inherits from GOPluginAction and defines the actual behavior of the plugin. The name of this class in order to be identified as an action class is required to be defined within *Plugins.xml* file in **GOPluginProjectWin8 (Windows 8)/GOPluginProjectWP8 (Windows Phone 8)** as described later in this document.

Within this class, the developer should provide the implementation in C# for the following method:

**Windows8\Windows Phone 8**

public override void execute(string action, string parameters, string callback)

{

base.execute(action, parameters, callback);

//User code

}

This method is intended to contain the logic of the action. Three parameters are provided, enabling the user to have access to the associated GOPlugin object, a string passing the specific plugin action, a string with the action parameters passed and a string representing the callback action.

* *action* provides the specific action to be performed.
* *parameters* provides parameters that define the specific actions to be performed.
* *callback* provides a string with the callback action to be executed in JavaScript defined in file  *index.html.*

**Notices**

*1) The developer must override the execute method of the GOPluginAction class and then depending on the value of the action parameter, the plugin's functionality can be called.*

*2) This method is called from the plugin SDK through a background thread. So if the plugin has code that needs to execute through the UI thread it must be enclosed inside as shown below*

**Windows 8**

Windows.ApplicationModel.Core.CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>

{

//User code

}

**Windows Phone 8**

System.Windows.Deployment.Current.Dispatcher.BeginInvoke(() =>

{

//User code

}

A method is available to be invoked by the developer:

**Windows8\Windows Phone 8**

public void executeCallback(string args)

This method enables the developer to invoke a callback function in JavaScript that is going to be executed within the current AppZone WebView container.

*The plugin developer should be aware that the execution of the action does not take place on the main thread.*

GOPluginProjectWin8 (Windows 8)/GOPluginProjectWP8 (Windows Phone 8)

The **GOPluginProjectWin8 (Windows 8)**/**GOPluginProjectWP8 (Windows Phone 8)** project provides the framework and the engine for developing, debugging and testing GOPlugins. This project’s source files and other resources are not included in GO!AppZone build services deployment, with the exception of *Plugins.xml* file.

In order to execute the plugin within the **GOPluginProjectWin8 (Windows 8)**/**GOPluginProjectWP8 (Windows Phone 8)**, the following steps need to be performed:

**Step 1**

The plugin should be registered as an entry in Plugins.xml file in the following way:

**Windows 8**

<plugin>

<assembly>PluginExampleWin8</assembly>

<name>Contacts plugin</name>

<class>PluginExample.ContactsPlugin</class>

<action>PluginExample.ContactsPluginAction</action>

<startup>onDemand</startup>

</plugin>

**Windows Phone 8**

<plugin>

<assembly>PluginExampleWP8</assembly>

<name>Contacts plugin</name>

<class>PluginExample.ContactsPlugin</class>

<action>PluginExample.ContactsPluginAction</action>

<startup>onDemand</startup>

</plugin>

The field ‘name’ should contain the name of the plugin. Field ‘class’ should contain the accompanying class that defines the plugin and should be created within the PluginExample subproject. The field ‘action’ should contain the class that defines the plugin action code. Finally, the field ’startup’ can take one of two values ’onStartup’ or ‘onDemand’. The value ‘onStartup’ denotes that the plugin should be initialized during the GO!AppZone launch, while ‘onDemand’ denotes that initialization should take place on the plugin’s first use.

**Step 2**

A JavaScript function should be provided in index.html file in "HTML" folder in order to test the intended action. This function should be responsible for initiating the action within the GO!AppZone application. See the following example:

JavaScript

function getContacts() {

go.executePlugin("Contacts plugin", "getContacts", "", "function(response){ alert(JSON.stringify(response))}");

}

The above function is pointing the GO!AppZone web container which in effect provides instructions regarding the plugin action to be executed.

The name of the JavaScript method that is executed is *go.executePlugin* which is directing GO!AppZone to start executing a plug in action. This method has four input parameters, *plugin,action,args, and callback* and defines the particular plugin which we intend to activate. The key plugin refers to the plugin class *e.g. ContactsAction*, while the key action refers to the action to be performed *e.g. getContacts*. The key args defines any arguments that must be passed to the plugin. The key callback defines the callback action in JavaScript, and it should be URL encoded. In case no arguments or callback action is needed should be equal with string empty.

***Notices*:**

**1)** The namespace to be used in a new plugin project **MUST NOT** start with "*GOEnterprise*".

This word is reserved and if it is used may create conflicts.

**2)** User can add to plugin project a library or package if is it is needed. But there are some restrictions to be noticed.

**Notice 1:** if there are any libraries (.dll) as a reference, the dll file must exist under plugin project folder. It can't exist outside the plugin project.

**Notice 2:** the plugin should use only the packages that windows 8 or windows phone 8 client use with the same version. For example if client uses a package *newtonsoft.json* with version *6.0.6* the plugin should use the same version for the package *newtonsoft.json*. If plugin has at least a package that client dont use the build of the application will fail.

**3)** Packages must be included at solution that will be given to the GO!Studio

The list with the packages that the client has, is:

**Windows 8**

<packages>

<package id="BugSense.W8" version="3.4" targetFramework="win" />

<package id="Callisto" version="1.3.1" targetFramework="win" />

<package id="Microsoft.Bcl" version="1.1.10" targetFramework="win" />

<package id="Microsoft.Bcl.Build" version="1.0.21" targetFramework="win" />

<package id="Microsoft.Net.Http" version="2.2.29" targetFramework="win" />

<package id="MimeKitLite" version="1.0.7.0" targetFramework="win" />

<package id="Newtonsoft.Json" version="6.0.8" targetFramework="win" />

<package id="winrtxamltoolkit" version="1.6.1.3" targetFramework="win" />

<package id="ZXing.Net" version="0.14.0.1" targetFramework="win" />

</packages>

**Windows Phone 8**

<packages>

<package id="BugSense.WP8" version="3.5.0.2" targetFramework="wp80" />

<package id="Microsoft.Bcl" version="1.1.10" targetFramework="wp80" />

<package id="Microsoft.Bcl.Async" version="1.0.168" targetFramework="wp80" />

<package id="Microsoft.Bcl.Build" version="1.0.21" targetFramework="wp80" />

<package id="Microsoft.Net.Http" version="2.2.29" targetFramework="wp80" />

<package id="Newtonsoft.Json" version="6.0.8" targetFramework="wp80" />

<package id="SharpZipLib-WP7" version="0.86.0.518" targetFramework="wp80" />

<package id="ZXing.Net" version="0.14.0.1" targetFramework="wp80" />

</packages>