Interaction Client Add-In

Quick Reference Guide

The Interaction Client Programmable Add-In API is used to customize and extend the functionality of the desktop Interaction Client.

Add-In Use Cases

- Custom Screen pop handler
- Adding a new tab in the client view
- Placing Calls
- Watching the user's queue
- Secure Input

First Steps to Create an Add-In

1. In Visual Studio, create a class library for the add-in. Make sure all add-in classs are public otherwise the client will not be able to load them.
2. In the Visual Studio project, add a reference to the ININ.InteractionClient.AddIn.dll file located in your Interaction Client directory. (Make sure to set the CopyLocal property to false)
3. In the Assembly.cs file, in the Assemblies section, add this line

```
[assembly: AddInVersion(AddInVersion.CurrentVersion)]
```

Queue Monitor Add-In

Queue monitor add-ins allow you to respond to interaction events (added, removed, changed) on a queue. Use the QueueMonitor base class to assist with setting up the watches and then override the base methods you need to use. Override the Attributes property to specify the interaction attributes that you want to receive events for. If there are interaction attributes that you need to access in an event handler but don't want to receive notifications when they change you can use the SupportingAttributes property.

```
public class AddIn : QueueMonitor
{
    protected override IEnumerable<string> Attributes
    {
        get
        {
            return new string[]
                {
                    InteractionAttributes.State
                };
        }
    }
    protected override void OnLoad(IServiceProvider serviceProvider)
    {
        base.OnLoad(serviceProvider);
    }
    protected override void OnUnload()
    {
        base.OnUnload();
    }
    protected override void InteractionAdded(IInteraction interaction)
    {
        base.InteractionAdded(interaction);
    }
    protected override void InteractionChanged(IInteraction interaction)
    {
        base.InteractionChanged(interaction);
    }
    protected override void InteractionRemoved(IInteraction interaction)
    {
        base.InteractionRemoved(interaction);
    }
}
```

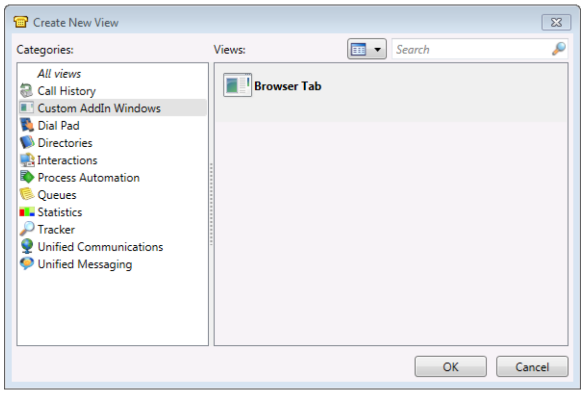
Client Tab Add-In

Add-Ins that create new client tabs can display custom WPF or WinForms controls in the Interaction Client. When creating a new tab, use the AddInWindow base class and implement the abstract properties on it.

CategoryDisplayName	The friendly name of the category. This is displayed in the Client Pages dialog.
CategoryId	The unique identifier of the window's category. If you are adding multiple custom windows and want them to appear in the same category, this value must match for each window.
Content	The user control or custom control to embed (docked to fill) inside the window when the window is created
DisplayName	The friendly name of the window. This is displayed in the Client Pages dialog when the user is selecting which pages to display in the client.
Id	The unique identifier of this window. This is used, for example, when the Interaction Client persists the open windows (tabs) during shutdown and re-creates each window on startup.

```
public class CustomClientWindow: AddInWindow
{
    private MyCustomControl _content;

    public CustomClientWindow()
    {
        _content = new MyCustomControl();
    }

    protected override string CategoryDisplayName
    {
        get { return "Custom AddIn Windows"; }
    }

    protected override string CategoryId
    {
        get { return "CustomAddInWindowsId"; }
    }

    public override object Content
    {
        get { return _content; }
    }

    protected override string DisplayName
    {
        get { return "Custom Control Tab"; }
    }

    protected override string Id
    {
        get { return "CustomControlTabID"; }
    }
}
```

The new window can be added to the client view by going into **File -> New View**. As of CIC 4.0 SU 4, it is not possible to customize the image shown in the New View Dialog



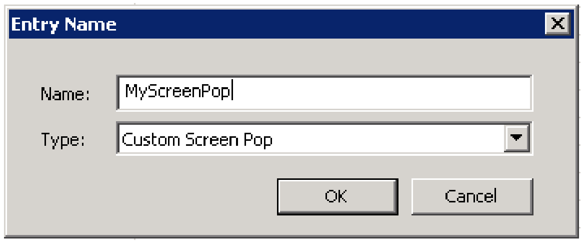
Screen Pop Add-In

Screen pop add-ins can leverage the screen pop functionality built into the IVR and Interaction Administrator to provide data to the screen pop add-in. When implementing a screen pop, you can use the ScreenPop base class and then override the Name property and Pop method. The Pop method will be called when an ACD interaction is alerting the user.

```
public class ScreenPopSample : ScreenPop
{
    public override string Name
    {
        get { return "MyScreenPop"; }
    }

    public override void Pop(IDictionary<string, string> attributes)
    {
        throw new NotImplementedException();
    }
}
```

Configuration in IA

A custom screen pop action will need to be setup in Interaction Administrator. In Interaction Administrator, go to the Actions Node under System Configuration and create a new Action. When creating the screen pop, the Type must be **Custom Screen Pop**

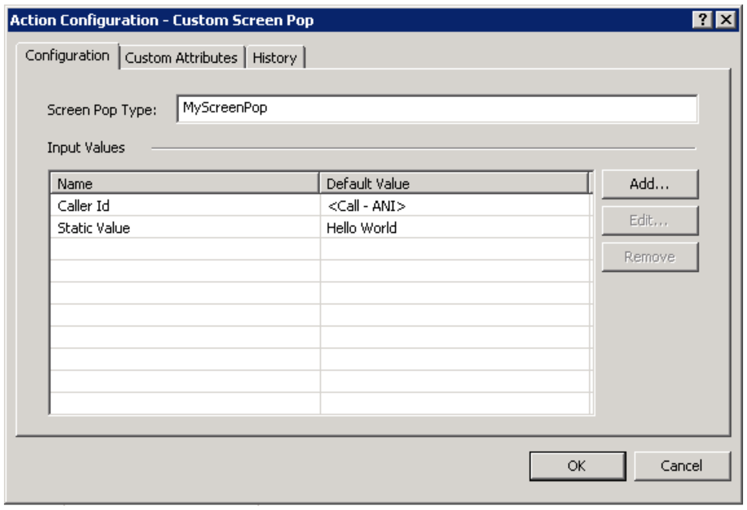


Name: MyScreenPop

Type: Custom Screen Pop

When editing the properties of the screen pop configuration, the input values are what will get passed into the Pop method in your Add-in as the attributes dictionary.

The Screen Pop Type of the custom screen pop action must match the Name property in your screen pop Add-in



Screen Pop Type: MyScreenPop

Name	Default Value
Caller Id	<Call - ANI>
Static Value	Hello World

Enabling the Screen Pop

The final thing that must happen is that the screen pop needs to be enabled for a call. This can be done by going to the Actions section of the ACD tab in a workgroup configuration and setting the Alerting Action to your newly created screen pop action. The other way is to use the Screen Pop node in Interaction Attendant to enable screen pop for any calls that flow through your IVR. This allows you to customize the data that is sent into the screen pop.

Deploying Your Add-In

An add-in is a simple DLL file that requires no other libraries and needs to be copied in the Program Files\Interactive Intelligence\ICUserApps\AddIns folder. The “AddIns” folder needs to be created if it is not there. If your add-in references other DLLs, they must also be copied in the same folder. If you end up referencing IceLib, be sure to not copy those files into the AddIns directory as they will already have been loaded by the Interaction Client

Use the following registry key value to determine the install location of the Interaction Client .NET edition:

HKEY_LOCAL_MACHINE\SOFTWARE\Interactive Intelligence\Installed\Interaction Client .Net Edition\InstallDir

Getting an IceLib Session in your Add-In

In order to be able to use the Session, you must have the IceLib license I3_FEATURE_ICELIB_SDK. Inside your Add-In, you have access to the IServiceProvider and as long as you have the IceLib license, you can get the session out of the service provider.

```
protected override void OnLoad(IServiceProvider serviceProvider)
{
    var session = (Session)serviceProvider.GetService(typeof(Session));
}
```

Make sure that in your visual studio project that you are NOT copying the IceLib dlls to the Addins folder. Doing so will cause a duplicate, and possibly different version of the dlls from what the client uses and can cause the .GetService method call to return null.

www.genesys.com