Workspace Desktop Edition Developer's Guide

Write Custom Applications

4/9/2025

Contents

Write Custom Applications

	Purpose: To provide information on how to implement the basic functions that are needed to write and deploy simple customized applications for Interaction Workspace.

Using Interaction Workspace API

You can use the Interaction Workspace API to write your custom-built .NET applications. After you have reviewed the information in this section, it might be useful to refer to the Extension Samples, upon which this document was based. The samples are working applications which contain and execute the functionality outlined here. When you are ready to write more complex applications, refer to the classes and methods described in the Interaction Workspace API Reference.

The Interaction Workspace API Reference is available in the installation package of the Interaction Workspace.

Basically, the API provides you with several managers which enables you to register your customized interfaces and behaviors:

- IObjectContainer-manages mapping of types and components (the same as IUnityContainer).
- IViewManager-manages the references on out-of-the-box views and regions.
- ICommandManager-manages the commands of your applications.
- IViewEventManager-manages the events launched by the views.

Through these managers, you can handle components detailed in Views and Regions to create views, regions, commands, or replace the existing objects. Additional manager interfaces provide access to the modules of the Interaction Workspace. You can find these interfaces in the feature's associated namespace. For instance, you can retrieve the `Genesyslab.Desktop.Modules.Core.Model.Broadcast.IBroadcastManager` interface as follow:

[C#]

```
public MyNewSampleClass(IObjectContainer container, ILogger log)
{
 IBroadcastManager broadcastManager = container.Resolve<IBroadcastManager>();
 //...
}
```

Because the Interaction Workspace is agent-oriented, the `Genesyslab.Desktop.Modules.Core.Model.Agents.IAgent` interface is also an important entry point, which can be retrieved in the same way:

[C#]

```
public MyNewSampleClass(IObjectContainer container, ILogger log)
{
 IAgent myAgent = container.Resolve<IAgent>();
 //...
}
```

Creating a New Module

The module is a top level component that can be loaded in the Interaction Workspace. The Interaction Workspace application is a modular application, so Genesys recommends that you create a new module which manages your customized behavior or appearance. The module can register and add new custom views and regions, or manage commands. The following steps explain how to create your new module which will handle the customized components. It is based on the `Genesyslab.Desktop.Modules.ExtensionSample`, which contains the new class `ExtensionSampleModule`. The `ExtensionSampleModule` class inherits from the `IModule` interface.

1. Create a new assembly `.dll`, and then a class that inherits from the `IModule` interface. In the following code sample, the module keeps a reference on the main manager instances that enable the customization.
   [C#]

```
// File: ExtensionModule.cs
 namespace Genesyslab.Desktop.Modules.ExtensionSample
 {
  public class ExtensionSampleModule : IModule
  {
  readonly IObjectContainer container;
  readonly IViewManager viewManager;
  readonly ICommandManager commandManager;
  readonly IViewEventManager viewEventManager;
  public ExtensionSampleModule(IObjectContainer container,
  IViewManager viewManager,
  ICommandManager commandManager,
  IViewEventManager viewEventManager)
  {
  this.container = container;
  this.viewManager = viewManager;
  this.commandManager = commandManager;
  this.viewEventManager = viewEventManager;
  }
  public void Initialize()
  {
  // Use the Initialize method to register your code, as described in further sections
  // In this example, the default DispositionCodeView is replaced with the
 DispositionCodeExView
  container.RegisterType<IDispositionCodeView, DispositionCodeExView>();
  }
  }
 }
```

   The purpose of the current module is to replace an existing view with a new view (detailed below).

2. You must modify the `Modules` section of the `ExtensionSample.module-config` file to include your new module. This can be done in two ways:
   - **Method 1** You add the following content to the `modules` section:
     [XML]

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <configSections>
 <section name="modules"
 type="Microsoft.Practices.Composite.Modularity.ModulesConfigurationSection,
Microsoft.Practices.Composite" />
 </configSections>
 <modules>
 <module assemblyFile="Genesyslab.Desktop.Modules.ExtensionSample.dll"
 moduleType="Genesyslab.Desktop.Modules.ExtensionSample.ExtensionSampleModule"
 moduleName="ExtensionSampleModule">
 <dependencies>
 <dependency moduleName="WindowsModule" />
 </dependencies>
 </module>
 </modules>

</configuration>
```

- **Method 2** You can load the module according to a `task` presence in the `tasks` section, as shown here:
  [XML]

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <configSections>
 <section name="tasks"
 type="Genesyslab.Desktop.Infrastructure.Config.TasksSection,
Genesyslab.Desktop.Infrastructure" />
 <section name="modules"
 type="Microsoft.Practices.Composite.Modularity.ModulesConfigurationSection,
Microsoft.Practices.Composite" />
 </configSections>
 <tasks>
 <task name="InteractionWorkspace.ExtensionSample.canUse"
 clickOnceGroupsToDownload="ExtensionSample"
 modulesToLoad="ExtensionSampleModule" />
 </tasks>
 <modules>
 <module assemblyFile="Genesyslab.Desktop.Modules.ExtensionSample.dll"
 moduleType="Genesyslab.Desktop.Modules.ExtensionSample.ExtensionSampleModule"
 moduleName="ExtensionSampleModule"
 startupLoaded="false"/>
 </modules>
</configuration>
```

Customizing Views and Regions

To customize your application, you can also create customized views and regions as described in Customize Views and Regions. The list of covered use cases is the following:

- Replacing an Existing View
- Creating a New View

Customizing a Command

To customize your application, you can also create customized commands in the existing chains of commands or use existing commands in your views. Help about using the commands is provided in the Use Customizable Commands page, which also links all the available commands. The list of covered use cases is the following:

- Creating a Command
- Inserting a Command in a Chain
- Multiple Commands and Overlapping

Deploying Your Custom Module into the Genesys Out-Of-The-Box Application

To deploy your custom module in a Click-Once deployment environment:

1. Write your custom code by using the provided APIs.
2. Compile the project as an assembly.
3. Unit test the project using a Unit Test framework or a simple test application.
4. Package the custom assembly with the Genesys out-of-the-box Interaction Workspace (by using Deployment Manager if Click-Once is used, or by using a custom IP).
5. In the Interaction Workspace application in Management Framework, configure a mapping between any custom tasks and the custom assembly.
6. Use the Genesys RBAC model to assign the custom task to users or to an access group of users.
7. Restart the Genesys Interaction Workspace.

For all other types of deployment, use the following steps to load and execute your custom assembly in Interaction Workspace:

1. Add your .dll file to the same directory as the one containing the `InteractionWorkspace.exe` file.
2. Create a new <`module`> section in the `ExtensionSample.module-config` file. See the examples in the previous section Creating a New Module.

Read Next

Customizing Work Items