UI Developers' Guide - Plugin Architecture

This documentation provides detailed information about the creation of plugins for the Ed-Fi Dashboards, including how plugins interact with data, services, and views, plus the overall plugin architecture and major conventions. This documentation also includes sample plugin implementations. The information provided should enable an accomplished developer to successfully leverage sample plugins, and to create new plugins for their Ed-Fi Dashboard projects.

The Ed-Fi platform architecture uses a collection of design patterns, architectural best-practices, conventions, and databases. In order to understand the code base and write a plugin it is important to understand these patterns and techniques (covered in the Architecture Detail section of this documentation). It is recommended that a developer intending to implement a plugin have knowledge and a reasonably solid understanding of the dashboard's extension mechanisms.

Basic Guidelines for Implementing a Plugin

There are a few fundamental guidelines that are important and should be followed when developing a plugin:

1. Plugins are mutually exclusive and self-contained. Plugins are intended to be new and additive functionality.
2. Plugins follow the basic Ed-Fi conventions for extensions. See the Extension Framework Guide f or details.
3. Plugins should use the convention-based project names starting always with EdFi.Dashboards. Plugins.[PluginName].[ProjectType]. For example, EdFi.Dashboards.Plugins.[PluginName].Resources, or EdFi.Dashboards.Plugins.[PluginName]. Data.
4. Plugin code is developed in alignment with the current EdFi.Dashboards.Plugins.Web version. **T his means that if a project is MVC 3 with .NET Framework 4.0, you should use MVC 3 with .NET Framework 4.0 or the plugin will not work.**
5. Referenced JavaScript and CSS files are declared as Embedded Resources.

By following these basic guidelines, you can develop extended features for the Ed-Fi Dashboards with a minimum of coding.

Technical Overview

The plugin architecture hooks into the existing dashboards by leveraging the CastleWindsor Inversion of Control (IoC) container. When the container is created (during Application_Start), it looks in the ~ /Plugins directory for appropriately named DLLs. If it finds any plugins, it then uses reflection to create instances of classes that implement the IWindsorInstaller interface. The IWindsorInstaller classes are responsible for wiring up the dependency injection for the plugin.

Once the IoC container has been initialized, the dashboard initializes the MVC framework. During this initialization, the dashboard will again look in the ~/Plugins directory for appropriately named DLLs. If it finds any plugins, it then registers all of the custom routes, controllers, and views. If the plugin has embedded JavaScript or CSS files, it creates the Cassette bundles for those embedded resources.

At this point, the plugin is available to be consumed.

Plugin walk-through samples:

- Plugin Architecture - Hello World (Sample Plugin)
- Plugin Architecture - Teacher's Student Notes (CRUD Example)

Troubleshooting Plugins

Security Exception

Error message: You do not have access to: [Resource].

Resolution: Make sure the methods on your controller and service have the required parameters in the signature. For example, if writing a plugin at the StudentSchool Level, make sure you have the SchoolId and the StudenUSI. You should also ensure that your service method is attributed with CanBeAuthorizedBy and the required claims:

```
[CanBeAuthorizedBy(EdFiClaimTypes.ViewAllStudents, EdFiClaimTypes.
ViewMyStudents)]
```

The Dashboard Doesn't Find My View

Error message: System.InvalidOperationException: The view “Get” or its master was not found or no view engine supports the searched locations. The following locations were searched: [Locations]

Resolution: Make sure the namespaces are correct. EdFi.Dashboards.Plugins.[PluginName].Web.Areas.[area].Views.[controllerName].Get. Also make sure that you have enabled Razor Generator on the view.

Controller Not Found Error

Error message: ControlType returns null while opening the sample plugin page in Admin login.

Resolution: Check the PluginHelper.GetPluginInstallers() which is called by InversionOfControlContainerFactory.GetInstallers() to ensure that the Plugin's installers are getting passed to the call to Castle.Windsor.Install(params IWindsorInstaller[] installers). This triggers Castle Windsor to call EdFi.Dashboards.Plugins.HelloWorld.Web.Utilities.CastleWindsor.Installer.Install which will call EdFi.Dashboards.Presentation.Core.Plugins.Utilities.CastleWindsor.WebDefaultConventionInstaller<Marker_EdFi_Dashboards_Plugins_HelloWorld_Web>.Install (IWindsorContainer container, IConfigurationStore store) which will call Castle.Windsor.WindsorContainer.Install(params IWindsorInstaller[] installers) passing in the EdFi.Dashboards.Presentation.Architecture.CastleWindsor.ControllerInstaller<Marker_EdFi_Dashboards_Plugins_HelloWorld_Web> which will trigger Castle Windsor to call EdFi.Dashboards.Presentation.Architecture.CastleWindsor..ControllerInstaller<Marker_EdFi_Dashboards_Plugins_HelloWorld_Web>.Install which uses reflection to find and register the Plugin Controller Types with Castle Windsor.

Unable to load one or more of the requested types Error

Error message: Unable to load one or more of the requested types. Retrieve the LoaderExceptions property for more information.

Resolution: In Visual Studio, right-click on the solution and select "Manage NuGet Packages for Solution...". In the consolidate tab, ensure that there are no items that need to be consolidated. If so, upgrade packages so that all projects have the same version of the package in question. Once you have done that, re-run the solution.