ACT PLUGIN MANAGER

WhitePaper

Table of Contents

[Overview 3](#_Toc156918023)

[Interfaces Used and Meanings 3](#_Toc156918024)

[ACT\_PluginManager\_Interfaces – Library 3](#_Toc156918025)

[I\_Cached\_Assembly 3](#_Toc156918026)

[Properties 3](#_Toc156918027)

[Methods 3](#_Toc156918028)

[I\_Assembly\_Loader 4](#_Toc156918029)

[Properties 4](#_Toc156918030)

[Methods 4](#_Toc156918031)

# Overview

ACT Plugin Manager is the Base Library that will Run All ACT Plugin Based Tools and Core Applications that use plugins. This is the lowest level ACT library and can be used alone outside of the usage of the core library.

This package will introduce the usage of some of the core structures that the rest of the packages use including the core library. JSON, License Files, Interface Usage, Resource Pathing and Multi-Delivery DotNet Package Flow.

# public static class \_\_

## JSON Configuration (Plugins – Interfaces)

This Core class contains many common helpful methods and support properties.

### *Application\_Plugins* **\_Application\_Plugins** & *Application\_Plugins* **\_Local\_Plugins**

This holds all the loaded JSON data found in the following files.

1. Application\_Plugins.json
   1. Application / Dependant Library
2. Local\_Plugins.json
   1. Local Plugins used to override Plugin Manager Code

### *Supported\_Interfaces* **\_Supported\_Interfaces** & *Supported\_Interfaces* **\_Local\_Supported\_Interfaces**

This holds all the loaded JSON data found in the following files.

1. Supported\_Interfaces.json
   1. Application / Dependant Library Interfaces
2. Local\_Supported\_Interfaces.json
   1. Local Interfaces supported by Plugin Manager

### *bool* **HasValidPlugins { get { } }**

1. This returns true if any of the Application Plugins were loaded and match the supported interfaces that were loaded.
2. If either is not loaded this method returns false.

### *bool* **HasLoadedInterfaces { get { } }**

1. This returns true if the supported interfaces were loaded and have at least 1 supported interface defined.

### *bool* **HasValid\_LocalPlugins { get { } }**

1. This returns true if any of the Local Plugins were loaded and match the local supported interfaces that were loaded.
2. If either is not loaded this method returns false.

### *bool* **HasLoaded\_LocalInterfaces { get { } }**

1. This returns true if the local supported interfaces were loaded and have at least 1 supported interface defined.

### Helpful Properties and Methods

1. public static string NL
   1. Returns the proper newline depending on the operating system.
2. public static string BaseDirectory
   1. Returns the base path to the current executing location.
   2. This can be overridden by adding a file to [**c:\ACT\baselocations.json**]
      1. See [Appendix C](#_Appendix_C) for more information!
3. public static string ResourceDirectory
4. public static string PluginsDirectory
5. public static string LocalPluginsDirectory
6. public static void LogFatalError(string messageToLog, Exception ex)
7. public static void LogBasicInfo(string messageToLog) { }

# Interfaces Used and Meanings

## ACT\_PluginManager\_Interfaces – Library

The ACT PluginManager Interface Library defines the interfaces that can be overridden by Plugins. This is accomplished by creating a separate engine to reload itself when plugins are detected internally. Let’s look at the available interfaces.

### I\_Cached\_Assembly

#### Properties

* + - Guid? Assembly\_CacheID
      * Used to potentially preload known assemblies and classes. If this is not used the system will still work it will just use the existing configuration and preload the information saving some loadtime.
      * See The Assembly\_CacheID Dirty Details Section for detailed usage instructions.
    - string Full\_PluginDLL\_Path
      * This holds the full hard drive location to the specific assembly. This is the Full Path to the DLL should point to a directory.
    - List<Type> Assembly\_Types
      * This holds all the internal types in other words all the methods, properties, etc, this will allow access to the public or assessable classes that implement various interfaces.
    - string PluginName
      * DLL Name without the path.
    - Assembly Loaded\_Assembly
      * Actual Assembly Loaded into Memory. Used to create Class instances for running plugins.
    - DateTime DateLoaded
      * For long-running applications this will let you know how stale the Assembly is.
    - string PluginInfo
      * This is just a basic string representing about this Assembly/Interface Implementation.
    - Note: This might seem redundant and confusing however remember this is a plugin storage and caching engine that is also a plugin. That is why things like PluginInfo are confusing.

#### Methods

* + - void Init\_Plugin(string PluginDLL\_Path, string Plugin\_Name, Assembly AssemblyToAdd, Guid? assembly\_CacheID);
      * This is what is called when you want to Init this plugin and is used internally to facilitate distribution.
    - string GetVersion();
      * This should return a string in ACT version format. See Appendix A.
    - I\_Cached\_Assembly CreateNew\_Cached\_Assembly\_FromDLL(string PluginDLL\_Path, string Plugin\_Name, Assembly AssemblyToAdd, Guid? assembly\_CacheID);
      * This will Create itself from the information to the right.
    - int Add\_Assembly\_Types(List<Type> types);
      * You can add assembly types directly.

I\_Assembly\_Loader

#### Properties

* + - string PluginInfo { get; }

#### Methods

* + - I\_Cached\_Assembly Load\_From\_FilePath(string dllFilePath);

# Main Engine - Setup

**Note \*\* = Common Shared Path for All ACT Products**

## Important File Locations

* Base\_Path: The location of the DLL running the ACT Plugin Manager
* **\*\***Resource: {BASE\_PATH}\Resouces\
  + This folder is typically found in all ACT Based Applications and holds the data, code, and other items used in configuration and setup.
* **\*\***Common Documentation: {BASE\_PATH}\Resouces\Documentation\
  + This folder is typically found in all ACT Based Applications and holds the documentation for various apps, libraries, and other items.
* Plugin Manager SDK: {BASE\_PATH}\Resouces\Documentation\ACT\_PLUGIN\_MANAGER\
  + This folder typically holds the ACT\_PLUGIN\_MANAGER Documentation from the SDK Install.
* **\*\***Plugins: {BASE\_PATH}\Resouces\Plugins\
  + This folder holds the DLL’s and Code Packages (ACTCP) that can be used in the Plugin Manager Engine.
    - ACTCP – Read More in Appendix B.
* **\*\***LocalPlugins: {BASE\_PATH}\Resouces\LocalPlugins\
  + This folder holds plugins for the Plugine Manager Only. This only supports compied DLLs.
    - ACTPE\_LocalPlugins – Read More in Appendix B.

## Application Plugins

Application plugins can reference one of two things. The JSON file defines which plugins your application will use and how they will be used. Below are both and are spelled out in detail.

### The JSON file is located below.

{BASE\_PATH}\Resouces\Plugins\Application\_Plugins.json

### Application\_Plugins Class

The class that loads the file and stores the data for access within the engine to understand the requirements of the initial configuration.

* 1. You can add and change the defined plugins through various methods then save out the data and update the settings.
  2. This also allows you to create a user interface that allows mods to be dynamically loaded.

### JSON File Structure and Meanings

{

"about": "View the SDK for a comprehensive list of all Interface and interface paths",

"app\_name": "File Example",

"interface\_plugin\_dlls": [

{

"interface": "IVolt.Interfaces.I\_HitSomething",

"dll\_name": "customplugins.dll",

"version\_increment": 20,

"plugin\_name": "",

"specific\_path": "",

"priority": 1,

"continue\_execution\_chain": true,

"replace\_core\_code": false,

"arguments": [ "###APPLICATION\_NAME###", "###FRAMEWORK###", "Mark Alicz" ],

"depends\_on": [

{ "interface": "IVolt.Interfaces.I\_HitSomething" },

{ "dll\_name": "customplugins.dll" }

{ "version\_increment\_min": 16 }

]

}

]

}

# Main Engine Functionality and Process Flow

## CurrentCore<T>

CurrentCore<T> is the MAIN Method you will use after you get the configuration and plugins in the correct location. This method can be used anytime you want to get the current plugin that is being used for the specific Type, which must be an interface; So, T Must be an Interface.

### Primary Methods

* + public static T GetCurrent()

#### Example Usage

* + You have a game that defines the I\_Player Interface. Anytime you want to create an instance of the current Plugin all you do is call this method like this.
    - using ACT.PluginManager;
    - var \_iPlayerObj = CurrentCore<I\_Player>.GetCurrent();

# Appendix A

# Appendix B

## ACTPE Local Plugins

Local Plugins are limited to the following Interfaces.

* I\_Cached\_Assembly
* I\_Assembly\_Loader

They can be in one DLL or separate and you can have many versions. To tell the system what plugin, if any, to use.

# Appendix C

## Overriding BaseDirectories

By creating a file located and named [**c:\ACT\baselocations.json**] will allow you to define override settings for ANY application. The template file is in the following path [**SDKROOT\Resources\JSON\_Templates\Global\BaseLocations.json**].

It is a simple json file with only one array of objects that define the application.

### Sample JSON File

|  |
| --- |
| {  “locations”: [  {  “id\_or\_name":”ACT\_PLUGIN\_MANAGER”  “base\_location”:”default”  },  {  “id\_or\_name":”YOUR\_PROGRAM”  “base\_location”:”c:\\somelocation\\somesublocation\\”  }  ]  } |

### Examples