Prime Plugin User Guide

Version 5.3

January 2026

SINCE DAY ONE.

Chyron Prime Plugin User Guide • 5.3 • January 2026 • This document is distributed by Chyron in online (electronic) form only, and is not available for purchase in printed form.

Table of Contents

Introduction to Prime Plugin

The Prime Graphics Platform defines a plugin architecture that allows for custom behavior to extend the usability of the software using .NET classes created in Visual Studio. This includes the ability to define: custom objects and effects that can be added to scenes, editors to adjust values for these objects and effects, runtime logic for the objects and effects, custom application-wide settings for the plugin, and editors to adjust values for the application-wide settings.

Steps to Create Prime Plugin

- Create a new Class Library project in Visual Studio targeting the correct .NET Version for Prime Version
    - Prime 5.0: .NET 8 or before
    - Prime 4.10: NET 6 or before
- Add Project references to the Prime dll's from the installed Prime Location. For example, for 5.0 the default location is **C:\Program Files\ChyronHego\Prime 5.0.0** folder
    - ChyronHego.Prime.Plugin.Base
    - ChyronHego.Prime.Plugin
    - ChyronHego.Prime.Scene
    - ChyronHego.Framework.Application
    - ChyronHego.Enterprise.Infrastructure
- Implement a class that implements an IPlugin Interface
- Compile and Build the Plugin Dll
- Place the Plugin

The Prime software will attempt to load all libraries located in the Prime Settings Plugins folder. For each successfully loaded library, a plugin will be instantiated for each class that implements the IPlugin interface in the ChyronHego.Prime.Plugin namespace.

IPlugin Interface

Add a new class to the project and have it implement the IPlugin interface. See the SamplePlugin.cs class in the SamplePlugin project for this example. The IPlugin interface defines 3 members that must be implemented:

None

```
using System.Collections.Generic;
using ChyronHego.Prime.Plugin;
using ChyronHego.Prime.Plugin.Settings;

public class SamplePlugin : IPlugin
{
    public string Name => "Sample Plugin"

    public IEnumerable<PluginObjectDefinition> ObjectDefinitions =>
Enumerable.Empty<PluginObjectDefinition>();

    public IPluginSettings Settings => null;
}
```

- **Name** defines the name of the plugin.
- **ObjectDefinitions** defines a list of objects that can be placed within Prime scenes.
- **Settings** define an object that can be used to edit and save application-wide settings for the plugin. Name must return a valid string.
- **ObjectDefinitions** can return an empty list, and **Settings** can return null if not needed.

The Prime software will attempt to load all libraries located in the Settings Plugins folder if it exists. For each successfully loaded library, a plugin will be instantiated for each class that implements the IPlugin interface in the ChyronHego.Prime.Plugin namespace.

PluginObjectDefinition Struct

The **PluginObjectDefinition** struct defines the name, object type, editor type, and runtime type to be used with a plugin object:

```
None
public string Name { get; set; }
public Type ObjectType { get; set; }
public Type EditorType { get; set; }
public Type RuntimeType { get; set; }
```

Name must return a valid string and ObjectType must return the type of a valid plugin object. EditorType and RuntimeType can both return null if not needed.

ObjectType Definition

Prime supports two types of objects that can be saved within scenes: Effects and Resources. Effects can be added as children of Graphic objects, allowing for behavior that can alter the parent graphic in some way, while Resource objects can be added to the Resources section of the scene to perform changes on the scene as a whole. The main purpose of this object type definition is to define the various properties that should be saved with this object.

PluginEffectBase Abstract Class

To create a type that represents an Effect, add a class to the project that inherits from PluginEffectBase in the ChyronHego.Prime.Plugin.Objects namespace. For example, the SamplePlugin project defines a TimeZonePluginEffect class that inherits from PluginEffectBase, while also defining a custom TimeZoneId string property. This effect was designed to update a parent Text object to display the time of the time zone specified in the effect.

PluginObjectBase Abstract Class

To create a type that represents a Resource, add a class to the project that inherits from PluginObjectBase in the ChyronHego.Prime.Plugin.Objects namespace. For example, the SamplePlugin project defines an OffsetPluginObject class that inherits from PluginObjectBase, while also defining a custom MinutesOffset integer property. This object was designed to store

a minutes offset value that all TimeZonePluginEffect objects can look for, and if found, offset the time zone by the specified number of minutes.

For each defined PluginObjectDefinition, set the ObjectType property to the type of the created effect or resource class.

EditorType Definition

Each plugin object can define an optional editor type that will be displayed in the Properties section of the Prime Designer when the object is selected, and allows editing the values of the properties defined with the object. To create a plugin object editor, add a new User Control (Windows Forms) to the project. After the user control has been added, change the base type of the class file from UserControl to PluginEditorBase defined in the ChyronHego.Prime.Plugin.Editor namespace. Note: this base class already inherits from UserControl. Before this editor is shown for the defined plugin object, the PluginObject property will be set with the instance of the selected plugin object. This property can be overridden to set up any necessary data binding for the designed property editors.

For example, the SamplePlugin project contains a TimeZonePluginEditor user control that inherits from PluginEditorBase, and displays a combo box that is populated with all known system time zones, and contains code to bind the TimeZoneId property of the TimeZonePluginEffect class to this combo box.

For each defined PluginObjectDefinition, set the EditorType property to the type of the created user control that corresponds with the object type.

RuntimeType Definition

Each plugin object can define an optional runtime class that will be instantiated when a scene is loaded on a playout channel. To create a runtime class, add a new Class to the project, and set the class to inherit from PluginRuntimeBase defined in the ChyronHego.Prime.Plugin.Runtime. This class provides a PluginObject property that can be overridden to access the plugin object that should be operated on. It also defines Load, Play, Stop and Clear methods that can be overridden to provide custom runtime behavior for the plugin object.

For example, the SamplePlugin project contains a TimeZonePluginRuntime class that provides behavior to update the parent text object of the associated TimeZonePluginEffect with the current time for the time zone of the effect's property.

IPluginSettings Interface

To define application-wide settings for the plugin, a class type can be specified for the Settings property of the main plugin class. This class must implement the IPluginSettings interface in the ChyronHego.Prime.Plugin.Settings namespace with the following properties:

None

```
string Name { get; }
Image Image { get; }
Type EditorType { get; }
```

Name and Image defines the name and image to be displayed in the application settings item list on the left hand side. EditorType defines the type of user control to be created to edit the settings.

ABOUT US

Chyron is ushering in the next generation of storytelling in the digital age. Founded in 1966, the company pioneered broadcast titling and graphics systems. With a strong foundation built on over 50 years of innovation and efficiency, the name Chyron is synonymous with broadcast graphics. Chyron continues that legacy as a global leader focused on customer-centric broadcast solutions. Today, the company offers production professionals the industry's most comprehensive software portfolio for designing, sharing, and playing live graphics to air with ease. Chyron products are increasingly deployed to empower OTA & OTT workflows and deliver richer, more immersive experiences for audiences and sports fans in the arena, at home, or on the go.

CONTACT SALES

EMEA • North America • Latin America • Asia/Pacific
+1.631.845.2000 • sales@chyron.com

Chyron