PRIME API and Scripting User Guide

Version 5.3

January 2026

SINCE DAY ONE.

Chyron PRIME API & Scripting 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

C# API

Introduction

PRIME versions 4.0 and later introduced support for C# scripting, which includes the ability to interact with and control the state of scenes, their actions and control panel controls through C# scripts. This document will be describing the PRIME version and scripting interface.

There is only one PRIME application running on a system at a time. The application services request through the API, providing the ability to retrieve and modify the state of scenes in the local PRIME database.

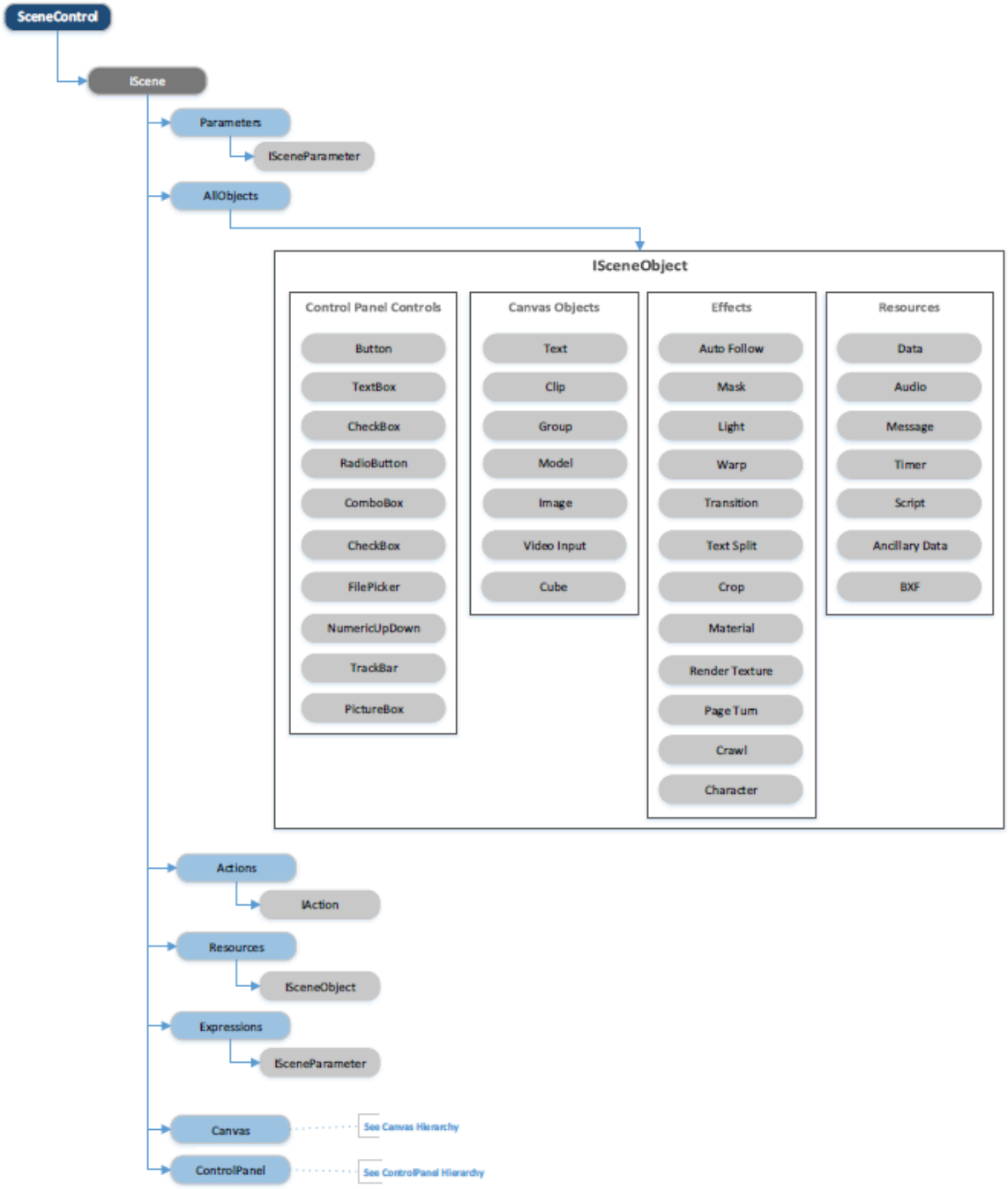
The **Scene** is the basic PRIME structure, containing multiple graphical, hardware and control elements called **SceneObjects** that define an interface to configurable aspects of the scene. Scenes also contain one or more **Actions**, which define object animations, and may be triggered through the scripting API. Each scene also has a collection of customizable **User Parameters**, which allows scripts to store and retrieve context information at runtime.

PRIME uses the Microsoft C# compiler to provide support for C# scripting. Scripts associated with a scene at design time are executed in-process. In-process scripts are provided context and a handle on the PRIME API in the form of global keywords that are accessible from the script. Other classes needed for more complex scripting are available through both Microsoft.NET assemblies as well as **PRIME** assemblies.

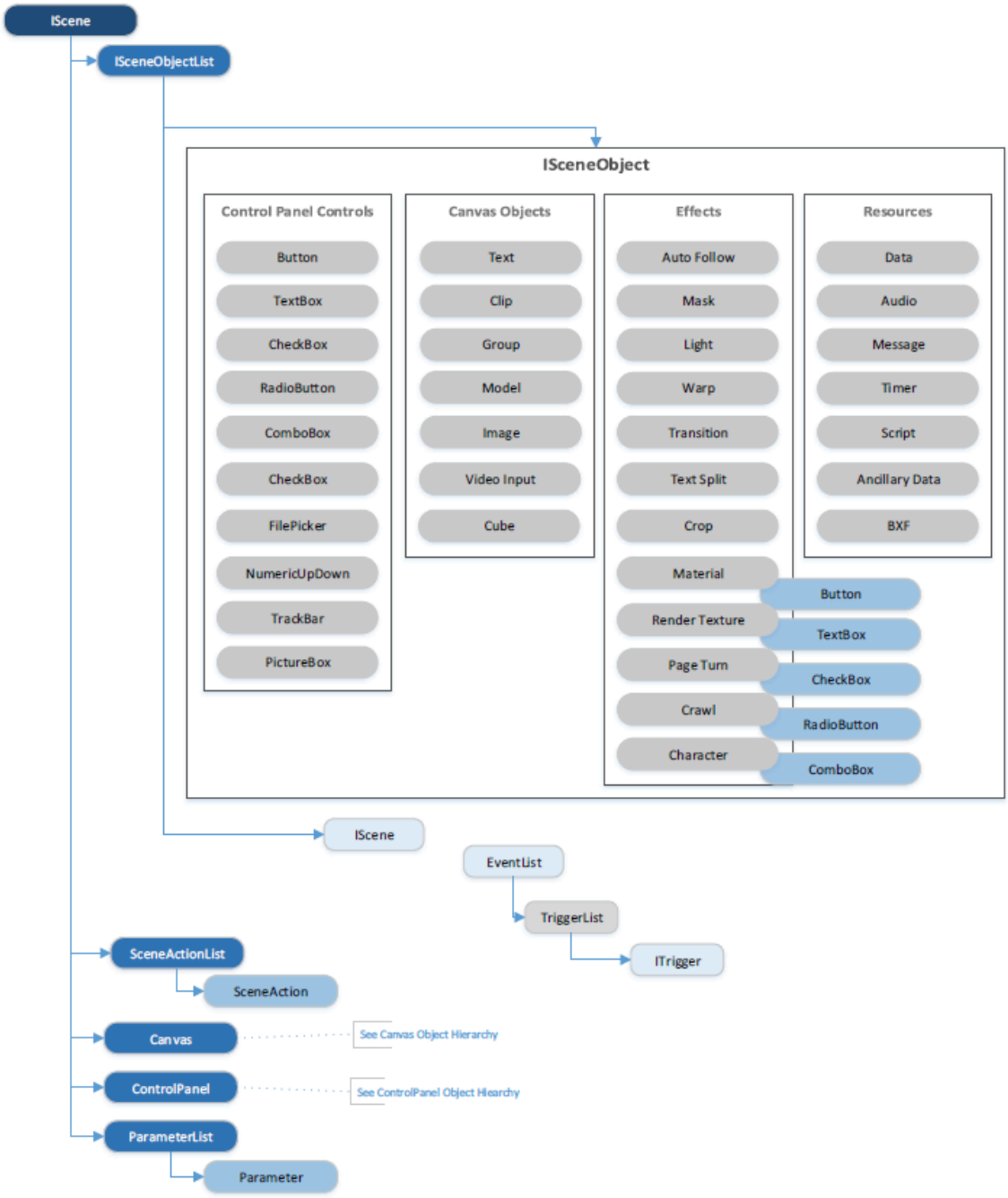
As of PRIME 4.5, the underlying architecture of PRIME was upgraded from .NET Framework to .NET6. Microsoft has changed the API in some cases and deprecated some functionalities. Therefore any customer upgrading to PRIME 4.5 or higher using C# scripting will need to retest their C# script and may need to make changes to their code if it directly makes calls to .NET library methods in order to make it compatible with .NET 6.

Object Hierarchy

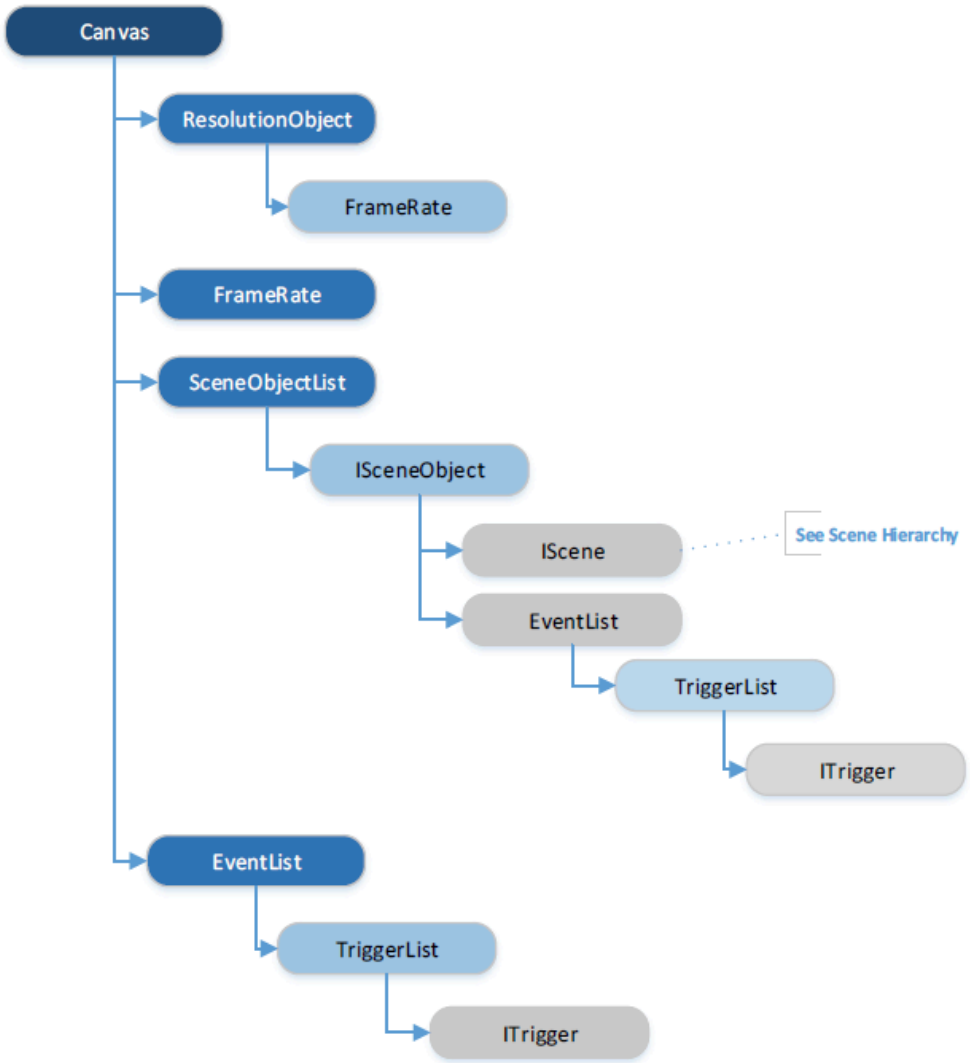
SceneControl Object Hierarchy



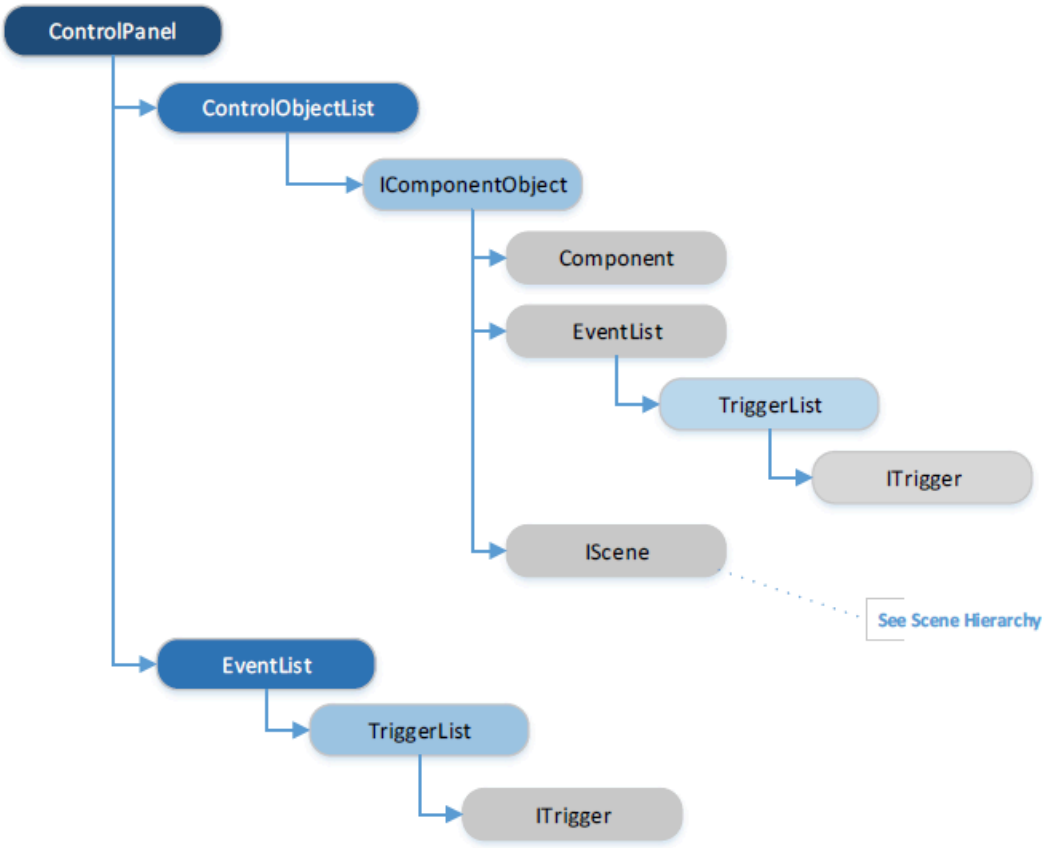
Scene Object Hierarchy



Canvas Object Hierarchy



ControlPanel Object Hierarchy



Scripting Objects

IScene

The scene object is available through the main script object, which is a **SceneControl**. Each scene contains a collection of graphical, hardware and control elements called **ISceneObjects**,

such as crawls, images and controls, which are exposed to scripting using the *AllObjects* enumeration properties, as shown below.

Namespace: **Chyron.PowerBox.Scene.Objects**

Properties

Name	Type	Description
Name	string	Gets the scene name.
Resolution	Resolution	Gets or sets the resolution of the scene (see **Resolution** section).
FrameRate	FrameRate	Gets or sets the frame rate of the scene (see **FrameRate** section)
Description	string	Gets the scene description.
Closed	bool	Gets a flag indicating if the scene is closed.
Playing	bool	Gets a flag indicating if the scene is playing.
Stopped	bool	Gets a flag indicating of the scene is stopped.
PlayoutState	PlayoutState	The **PlayoutState** is an enumerated value that indicates the current state of the scene. Below is a list of valid values and what they mean. *Closed* Scene in not in preview of on air. *Loaded* Scene is in preview *Playing* Scene is on air. *Stopping* Scene is being stopped (being transferred from on air to preview) *Stopped* Scene in in preview.
Status	string	Gets a descriptive status of the scene.
FilePath	string	Gets the file with full path of the scene.
Directory	string	Gets the directory of the scene.
ProjectDirectory	string	Gets the directory of the project the scene is contained in.
AllObjects	IEnumerable<ISceneObject>	Get the list of all objects including graphics, control panel, resources and effects that the scene contains.
Actions	SceneActionList	Gets the list of Actions defined for this scene.
Canvas	Canvas	Gets the canvas object used for the display of graphics and effects.
Scripting	Scripting	Gets the scripting object for this scene.
ControlPanel	ControlPanel	Gets the control panel object used in the display of the control panel objects.
Resources	SceneResources	Gets the list of resources defined for this scene such as DataObjects,
Parameters	ParameterList	Gets the list of parameters defined for the scene.
Expressions	ExpressionList	Gets the list of expressions defined for the scene.

Events

PlayoutStateChanged

The event is triggered when the state of the scene changes.

Example:

```
public ScriptTest()
{
      InitializeComponent();

      // Hookup your handler to the scenes event
      this.Scene.PlayoutStateChanged += OnPlayoutStateChanged;

}


// Handles the PlayoutStateChnaged event from the scene
private void OnPlayoutStatedChanged(IScene scene)
{
  switch ( scene.PlayoutState )
  {
    case PlayoutState.Loaded:
      // You code to handle the Loaded state
      break;
    case PlayoutState.Playing:
      // Your code to handle the Playing state
      break;
    case PlayoutState.Closed:
      // Your code to handle the Closed state
      break;
    case PlayoutState.Stopped:
      // Your code to handle the Stopped state.
      break;
   }
}
```

ActionPlayed

The event is triggered when one of the scenes actions is played.

Example:

```
public ScriptTest()
{
      InitializeComponent();

      // Hookup your handler to the scenes event
      this.Scene.ActionPlayed += OnActionPlayed;

}


// Handles the ActionPlayed event from the scene
private void OnActionPlayed(IAction action)
{
      // Your code to handle the ActionPlayed event
}
```

Methods

FindObject

Find the specified object by name from the list of objects the scene contains. This includes graphical, control panel, resource and effect objects. If the object for the name specified was not found, then null is returned so a check of the return value is necessary.

Syntax:

```
ISceneObject FindObject(string name)
```

Example:

```
// Attempt to find object “MyCrawl” in scene
var effect = Scene.FindObject("MyCrawl");
if ( effect != null )
{
      // Add you code here
}
else
{
      Scene.LogError("MyCrawl not found")
}
```

LogError

Logs the specific string to the log file.

Syntax:

```
void LogError(string error)
```

Example:

This example tries to find and scene object called “MyObject” in the list. If the object is not found the value returned is null and is checked so that a message can be logged.

```
// Attempt to find object “MyObject” in scene
var obj = Scene.FindObject("MyObject");
if ( obj != null )
{
      // Add you code here
}
else
{
      Scene.LogError("MyObject not found")
}
```

LogException

Log the specified exception to the log file.

Syntax:

```
void LogException(Exception exception)
```

Example:

The **LogException()** method is used in conjunction with try-catch construct seen below. You wrap you code in a try catch block in case an exception is thrown so can log it as well as have the ability to handle it gracefully.

```
private void MyMethod()
{

	try
	{
		// Add your code here
	}
	catch ( Exception ex )
	{
		Scene.LogException(ex);

	}
}
```

ISceneObject

The **ISceneObject** is the interface of all Control Panel, Canvas and Resource and Effect objects in a scene. The **ISceneObject** allows a script author the ability to affect the state of any objects that the scene contains. To access the **ISceneObjects** from a **Scene** you would use either **AllObjects** property or the **FindObject()** method.

Namespace: **Chyron.Framework.Scene.Objects**

Properties

Name	Type	Description
Name	string	Gets the name of the scene object.
ScriptName	string	Gets the script name for the scene object.
Scene	IScene	Gets the scene the scene object is associated with.
Events	EventList	Gets the list of events associated with the scene object.
Parent	ISceneObjectParent	Gets the parent of the scene object.

Events

PropertyChanged

Event is triggered when one of the scene objects properties has changed.

Syntax:

```
void PropertyChanged(string property)
```

BeforePropertyChanged

Event is triggered prior to a property being changed allowing for the scene author to perform some processing prior to a particular property on a scene object is performed. Notice that the first parameter in the syntax identifies the **ISceneObject** that is triggering this event. This allows the scene author to use the same event handler to handle multiple **BeforePropertyChanged** events from different **ISceneObject's**.

Syntax:

```
void BeforePropertyChanged(ISceneObject sceneObject, string property, object
value)
```

AfterPropertyChanged

Event is triggered after the property has been changed allowing for the scene author to perform some processing after a particular property on a scene object is performed. Notice that the first parameter in the syntax identifies the **ISceneObject** that is triggering this event. This allows the scene author to use the same event handler to handle multiple **AfterPropertyChanged** events from different **ISceneObjects**.

Syntax:

```
void AfterPropertyChanged(ISceneObject sceneObject, string property, object value)
```

Example:

Register event handlers for the **PropertyChanged**, **BeforePropertyChanged** and **AfterPropertyChanged** event so that the scene author can perform processing when these events are triggered.

```
public ScriptTest()
{
	InitializeComponent();

	var textbox = this.Scene.FindObject("MyTextBox");

	if ( textbox != null )
	{
		textbox.PropertyChanged += OnPropertyChanged;
		textbox.BeforePropertyChanged += OnBeforePropertyChanged;
		textbox.AfterPropertyChanged += OnAfterPropertyChanged;

	}


}
```

Methods

Exists

Returns true if the property specified by type exists in the property list.

Syntax:

```
bool Exists(string property)
```

Example:

```
// Get the SAMPLE scene from the ChannelBox database
var scene = ChannelBox.GetScene("SAMPLE");

// Gets the object MyTextBox from the SAMPLE scene
var obj = scene.FindSceneObject("MyTextBox");

// Check if the MyTextBox has a Text property that has been set. If true
// then access the value otherwise ignore since we don't want use
// the default value.
if (obj.Exists("Text"))
{
     // Get the Text property value
     var value = obj.GetValue("Text");

     // Do something with the value
 }
```

GetValue

Returns the value of the property specified if it exists otherwise a **null** is returned.

Syntax:

```
object GetValue(string property)
```

Example:

Get the "MyTextBox" scene object and make sure that it was found by checking against the null value. If the scene object was found, then try to get the value for "PropertyName". Check the value return against null to make sure the property was found.

```
var textbox = this.Scene.FindObject("MyTextBox");
if ( textbox != null )
{
      var value = textbox.GetValue("PropertyName");

      if ( value != null )
      {
            // Your code to use the value for PropertyName
      }
```

SetValue

Sets the property value for the *type* specified.

Syntax:

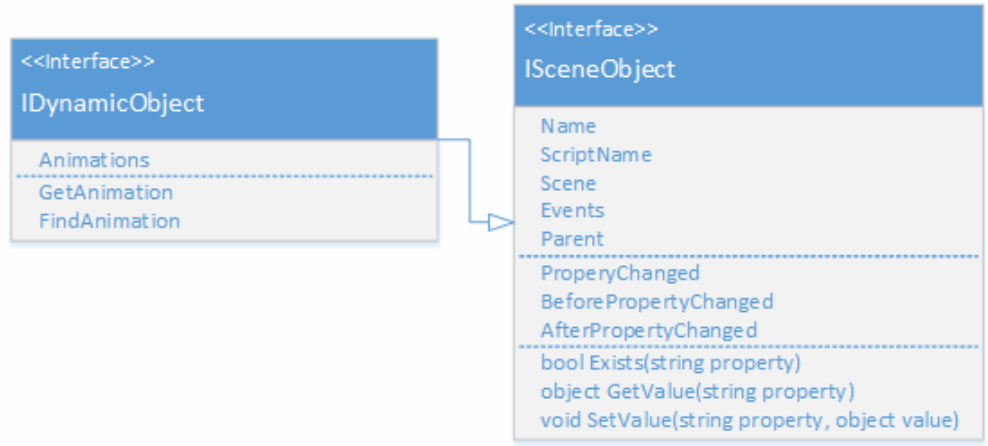
```
void SetValue(string property, object value)
```

Example:

Get the “MyTextBox” scene object and make sure that it was found by checking against the null value. If the scene object was found, then set the Text property of the scene object to “Test.

```
var textbox = this.Scene.FindObject("MyTextBox");
if ( textbox != null )
{
      textbox.SetValue("Text", “Test”);

}
```

IDynamicObject



The **IDynamicObject** is the base interface for graphics and effects scene objects. It extends the functionality of the **ISceneObject** described earlier.

Namespace: **Chyron.Framework.Scene.Objects**

Properties

Name	Type	Description
Animations	AnimationList	Gets the list of animations for this scene object.

Events

PropertyAnimated

Event is triggered when one of the scene properties is animated.

Syntax:

```
void PropertyAnimated(IDynamicObject parent, string property)
```

Example:

Register the **OnPropertyAnimated** event handler to a graphics Text objects (Text1) **PropertyAnimated** event. This example handles the property-animated event for the “MyGraphicText” object and x position property.

```
public ScriptTest()
{
      InitializeComponent();

      Text1.PropertyAnimated += OnPropertyAnimated;



}
// Event handler for the PropertyAnimated event of the scene object
private void OnPropertyAnimated(IDynamicObject dynamicObject, string
property,
                                    object value)
{
      if ( dynamicObject.Name == "MyGraphicText" && property == "PositionX" )
      {
            // Your code for handling the property animated event
      }
}
```

Methods

FindAnimation

Attempts to find the animation using the name specified and return it. If the animation was not found then a null is returned.

Syntax:

```
IAnimation FindAnimation(string name)
```

Example:

Attempt to get the animation using the name specified “MyAction” and if found then the animation object is returned. We must check for null before to make sure that a valid object was returned.

```
private void FindAnimationExample()
{

	var animation = Text1.FindAnimation("MyAction");

	if ( animation != null )
	{
		// Your code to manipulate the animation
	}

}
```

GetAnimation

Attempts to find the animation using the name specified and return it. If the animation was not found, then a new animation with the name specified is returned.

Syntax:

```
IAnimation GetAnimation(string name)
```

Example:

Attempt to get the animation using the name specified “MyAction”, if the animation is not found then a new animation is created and returned.

```
private void GetAnimationExample()
{
	var animation = Text1.GetAnimation("MyAction");
	// Your code to manipulate the animation
}
```

IAction

The object describes an action used within a scene. It is the element used within the SceneActionList described below.

Namespace: **Chyron.Powerbox.Scene.Actions**

Properties

Name	Type	Description
Name	string	Gets the name of the Action
ScriptName	string	Gets the name of the script associated with this Action
Length	TimeCode	Gets or sets the length in time for this Action
Animations	IEnumerable<IAnimation>	Gets the list of animations associated with this Action
Playing	bool	Gets the flag indicating if the Action is playing or not.

Events

Name	Description
Played	Event triggered when the action is played
Finished	Event triggered when the action has finished playing.

Methods

Play

Plays the action.

Syntax:

```
void Play()
```

Example:

Handles a SelectedIndexChanged event for a ComboBox (see Control Panel section) that takes the current selected action name from the MyActionComboBox and use that to lookup the action in the current scenes action list. If the action was found and is not playing, then play the action.

```
private void OnSlectedIndexChanged(object sender, EventArgs args)
{
	var name = MyActionComboBox.SelectedItem as string;

	var action = this.Scene.Actions.Find(name);

	if ( action !=- null && !action.Playing)
	{
		action.Play();
	}
}
```

Stop

Stops the action if being played.

Syntax:

```
void Stop()
```

Example:

Handles a SelectedIndexChanged event for a ComboBox (see Control Panel section) that will take the current selected action name from the MyActionComboBox and use that to lookup the action in the current scenes action list. If the action was found and is playing, then stop the action.

```
private void OnSlectedIndexChanged(object sender, EventArgs args)
{
	var name = MyActionComboBox.SelectedItem as string;

	var action = this.Scene.Actions.Find(name);

	if ( action !=- null && action.Playing)
	{
		action.Stop();
	}
}
```

Reverse

Reverses the action.

Syntax:

```
void Reverse()
```

Example:

Handles a SelectedIndexChanged event for a ComboBox (see Control Panel section) that will take the current selected action name from the MyActionComboBox and use that to lookup the action in the current scenes action list. If the action was found, and is not playing, then reverse the action.

```
private void OnSlectedIndexChanged(object sender, EventArgs args)
{
	var name = MyActionComboBox.SelectedItem as string;

	var action = this.Scene.Actions.Find(name);

	if ( action !=- null && !action.Playing)
	{
		action.Reverse();
	}
}
```

SceneActionList

Contains a list of **IAction** elements for a scene that allows for the access, manipulation and monitoring of these elements.

Namespace: Chyron.PowerBox.Scene.Actions

Properties

Name	Type	Description
Count	int	The number of **IAction** elements in the list.

Methods

Contains

Returns true if an action with the name specified exists in the list, false if it does not.

Syntax:

```
bool Contains(string name)
```

Example:

Check if the current scenes action list contains the action “MyAction” in the list prior to using the Find() method and manipulating the action.

```
private void ContainsActionExample()
{

      if ( this.Scene.Actions.Contains("MyAction")  == true )
      {
            var action = this.Scene.Actions.Find("MyAction");

            // Your code to manipulate the action
      }

}
```

Find

Returns the **IAction** element that matches the name specified if found or null is not found. A check of the return value against null is required.

Syntax:

```
IAction Find(string name)
```

Example:

Attempt to find “MyAction” action in the current scenes action list. Check to make sure that the action was found by checking the return value against null.

```
private void FindActionExample()
{

	var action = this.Scene.Actions.Find("MyAction");

	if ( action != null )
	{
		// Your code to manipulate the action
	}

}
```

Animation

This object describes an action used within a scene. It is the element used within the **SceneActionList** described below.

Namespace: **Chyron.PowerBox.Scene.Objects**

Properties

Name	Type	Description
Name	string	Gets the name of the Animation
ScriptName	string	Gets the script name for this animation.
Loop	bool	Gets the flag indicating whether the animation is to loop or not.
HasKeyFrames	bool	Gets the flag indicating whether the animation contains keyframes.
IsDefault	bool	Gets the flag indicating whether this animation is the default.
ParentObject	IDynamicObject	Gets the parent object that owns this animation.
Length	TimeCode	Gets the length in time for this animation.
Keyframes	KeyframeList	Gets the list of keyframes associated with this animation.

[string name]	IKeyframe	Indexer allowing the scene author access to the keyframes associated with this animation as if accessing an array of elements using the name. When using the indexer to access animations keyframes it works just like the **Find()** method described below in which case a null is returned if the name specified is not found.
		Example:
		`private void FindAnimationKeyframeExampleUsingIndexer()`
		`{`
		`  var animation =`
		`Text1.Animations.Find("MyAnimation");`
		`  if ( animation != null )`
		`  {`
		`  var keyframe =`
		`animation["MyStartingKeyFrame"];`
		`  if ( keyframe != null )`
		`  {`
		`  // Your code to manipulate the`
		`  // animations keyframe`
		`  }`
		`  }`
		`}`

[int index]	IKeyframe	Indexer allowing the scene author iterate over all the animations keyframes by using an indexer.

		Example:
		
		`private void`
		
		`FindAnimationKeyframeExampleUsingIndexer()`
		`{`
		
		`  var animation =`
		`  Text1.Animations.Find("MyAnimation");`
		
		`  if ( animation != null )`
		`  {`
		`  for ( var i = 0;`
		`  i < animation.Keyframes.Count;`
		`  i++ )`
		`  {`
		`  var keyframe = animation[i];`
		
		`  // Your code to manipulate the`
		`  // animations keyframe`
		
		`  }`
		`  }`
		`}`

Methods

Find

There are two methods available which enables the author to find a keyframe in the animation. One uses the name and the other the frame index. If any of these methods cannot find the keyframe based on the parameter then a null is retuned.

Syntax:

```
IKeyframe Find(string name)

IKeyframe Find(int frame)
```

Example:

Attempts to find the “MyKeyframe” keyframe in the “MyAnimation” animation. Notice that each find checks the returned value against null to make sure a valid object is returned.

```
private void FindKeyframeExample ()
{

	var animation = Text1.Animations.Find("MyAnimation");

	if ( animation != null )
	{
		var keyframe = animation.Find("MyKeyframe");

		if ( keyframe != null )
		{
			// You code to manipulate the keyframe
		}
	}

}
```

GetFirstKeyframe

Syntax:

```
IKeyframe GetFirstKeyframe();
```

Example:

Gets the first keyframe from “MyAnimation” animation. Notice that the keyframe returned from GetFirstKeyframe() is checked against null in case not keyframes are present.

```
private void GetFirstKeyframeExample ()
{
	var animation = Text1.Animations.Find("MyAnimation");

	if ( animation != null )
	{
		var keyframe = animation.GetFirstKeyframe();

		if ( keyframe != null )
		{
			// You code to manipulate the keyframe
		}
	}
}
```

GetLastKeyframe

Syntax:

```
IKeyframe GetLastKeyframe()
```

Example:

Gets the last keyframe from “MyAnimation” animation. Notice that the keyframe returned from GetLastKeyframe() is checked against null in case not keyframes are present.

```
private void GetLastKeyframeExample ()
{
	var animation = Text1.Animations.Find("MyAnimation");
	if ( animation != null )
	{
		var keyframe = animation.GetLastKeyframe();
		if ( keyframe != null )
		{
			// You code to manipulate the keyframe
		}
	}
}
```

AnimationList

Contains a list of **IAnimation** elements for a scene that allows for the access, manipulation and monitoring of these elements.

Namespace: Chyron.PowerBox.Scene.Objects

Properties

Name	Type	Description
Count	int	The number of **IAnimation** elements in the list.
AllKeyfarmes	IEnumerable<IKeyframe>	Returns a list of all keyframes from all the animations.

[string name]	IAnimation	Indexer allowing the scene author access to the animations as if accessing an array of elements using the name. When using the indexer to access animations it works just like the **Find()** method described below in which case a null is returned if the name specified is not found.
		
		Example:
		`private void FindAnimationExampleUsingIndexer()`
		`{`
		`var animation = Text1.Animations["MyAnimation"];`
		`if ( animation!= null )`
		`{`
		`// Your code to manipulate the`
		`// animation`
		`}`
		`}`

Methods

Find

Attempts to find the animation using the name specified. If the animation is found, then the object is returned otherwise a null is returned if the animation is not found.

Syntax:

```
IAnimation Find(string name)
```

Example:

Attempt to find “MyAnimation” animation in the graphics text object Text1. Check to make sure that the animation was found by checking the return value against null.

```
private void FindAnimationExample()
{
	var animation = Text1.Animations.Find("MyAnimation");

	if ( animation!= null )
	{
		// Your code to manipulate the animation
	}
}
```

Get

Attempts to find the animation using the name specified and it is does not then it will add a new animation with the name specified.

Note – Care should be taken if this method is used, since new animations will be added if the name specified is not found. The **Find()** method is the suggested method to use.

Syntax:

```
IAnimation Get(string name)
```

Example:

Attempt to find “MyAnimation” animation in the graphics text object Text1 if it is not found then a new animation is added with the name specified. Notice that no check is required for the animation returned as it is either found or created so a valid object will always be returned.

```
private void GetAnimationExample()
{
      var animation = Text1.Animations.Get("MyAnimation");
      // Your code to manipulate the animation

}
```

IParameter

The **IParameter** is interface used to define all parameter type objects with the PRIME system. It is used as the object type for the **IParameterList** described below, and is used to access, modify and manipulate parameter values within the scene.

Namespace - Chyron.Enterprise.Infrastructure.Parameters

Properties

Type	Name	Description
string	**Name**	Gets the name of the Parameter
object	**Value**	Gets or sets the value of the parameter

IParameterList

The **IParameterList** is the interface for all lists utilizing the **IParameter** interface described above. It allows for the access, modification and manipulation of these elements using the methods defined below.

Namespace: Chyron.Enterprise.Infrastructure.Parameters

Events

Name	Description
ParameterChanged	This event is triggered when an **IParameter** element in the list has changed. This event can be used to detect when a particular parameter of interest value has been modified so that some process can take place.

Example:

The handler must first be attached to the event such as in the constructor method, which is the same name as the scene, which is seen below, and is called when the main scripting object is created. The using line must be added to the list at the beginning of the script so that the **IParameter** properties and methods can be accessed in the handler.

```
// Required for IParameter type in handler
using Chyron.Enterprise.Infrastructure.Parameters;
// constructor
public ScriptTest()
{
	InitializeComponent();

	// Attach handler to event
	this.Parameters.ParameterChanged += OnParameterChanged;

}
	…

// Handles the ParameterChanged event
private void OnParameterChanged(IParameter parameter, object previousValue)
{
	if ( parameter.Name == "MyParameter" )
	{
		try
		{
			// Add your code here
		}
		catch ( Exception ex )
		{
			// Add you code to handle exception
			Log("Failed to handle ParameterChanged event");
		}
	}
}
```

Methods

IParameter Find(string name)

Attempts to find the **IParameter** element with the name specified in the list. If found, then the parameter is return otherwise a null value is returned. The return value should be checked for null to determine if the parameter was found and proper handling should be implemented.

Example:

Find a **IParameter** element by name to get the value. Verify that the parameter actually was found by checking the if the return value is null. If the parameter was not found log some error message and use a default value otherwise use the parameters value.

```
var parameterValue = "DefaultParameterValue";
var parameter = this.Parameters.Find("MyParameter");
if ( parameter == null )
{
  Log("MyParameter was not found");
}
else
{
  parameterValue = Convert.ToString(parameter.Value);
}
```

IParameter Set(string name, object value)

Sets or adds the value of the **IParameter** element whose name is specified in the method. If the **IParameter** element specified by the name already exists, then the value is set but if it not then a new **IParameter** element is added with name and values specified. Notice that the value of the parameter is a generic object which is to allow a parameter to be any one of the following types (Boolean, ByteArray, DateTime, Double, Int, Long, String and TimeSpan).

Example:

```
var parameterValue = "NewValue";
var parameter = this.Parameters.Set("MyParameter", parameterValue);
```

IParameter Add(object value)

Adds a new **IParameter** element to the list with the value specified. The name of the parameter will be “Parameter N” where N is an incremental number for the parameter. Notice that the value of the parameter is a generic object which is because a parameter can be any one of the following types (Boolean, ByteArray, DateTime, Double, Int, Long, String and TimeSpan). The parameter added will only be temporary and is only available during the playout.

Example:

```
var newParameter = this.Parameters.Add("This is a test");
```

void Remove(IParameter parameter)

Removes a **IParameter element** from the list such as a temporary one that was added with the Add() method described above.

Example:

```
this.Parameters.Remove(newParameter);
```

Warning:

Removing parameters that were not added through the Add() or Set() method is not recommended as it may cause adverse and/or unexpected affects during the playout.

SceneControl

SceneControl is the top-level object available to scene script. This object allows the script author the ability to affect the state of the current scene and its objects, store and retrieve context information for use by other scripts, and a plethora of other features.

Namespace: Chyron.PowerBox.Playout.UserInterface

Properties

Name	Type	Description
Scene	IScene	This returns the current scene that is being acted upon. Access to all scene properties are done through this property (see **Scene** section)
Parameters	IParameterList	This property allows the script author to access and modify any of the parameters defined in the scene (see **IParameterList** section)
InvokeRequired	bool	If this property is true then **Invoke()** method needs to be used update any controls on the Control Panel. If false then **Invoke()** is not required and update to any of the Control Panel controls directly.

Example:

To enable a **MyButton** button on the control panel the following code is needed if this is being done on a separate thread.

```
// Enable/Disable MyButton on control panel
private void EnableButton(bool enable)
{
	if ( this.InvokeRequired )
	{
		Invoke(new Action<bool>(EnableButton), new object [] { enable }
);
	}
	else
	{
		MyButton.Enabled = enable;
	}
}
```

Events

To utilize events within your script you will need to attach an event handler that will perform some task before continuing with the process. Take note that tasks within the event handlers should be kept short so as not to delay the scene. In addition, error handling should be considered, as any code that generates an exception will cause the scene to stop processing at that point.

BeforeLoad

This event is triggered before the scene is loaded into preview.

Syntax:

```
void BeforeLoad(IScene scene)
```

Example:

Attach your event handler method **OnBeforeLoad** to the **BeforeLoad** event in the constructor of the scenes script class. Any time the **BeforeLoad** event is triggered the **OnBeforeLoad** method will be called.

```
// constructor
public ScriptTest()
{
      InitializeComponent();

      // Attach event handler to event
      this.BeforeLoad += OnBeforeLoad;


}

...


// Handles the BeforeLoad event

private void OnBeforeLoad(IScene scene)
{
  // Add your code here, but make sure it is not a long process
}
```

AfterLoad

This event is triggered after the scene is loaded into preview. It will not be triggered if the scene is taken to air directly. This event is triggered before the scene is loaded into preview.

Syntax:

```
void AfterLoad(IScene scene)
```

Example:

Attach your event handler method **OnAfterLoad** to the **AfterLoad** event in the constructor of the scenes script class. Any time the **AfterLoad** event is triggered the **OnAfterLoad** method will be called.

```
// constructor
public ScriptTest()
{
      InitializeComponent();

      // Attach event handler to event
      this.AfterLaod += OnAfterLoad;

}

      …


// Handles the AfterLoad event

private void OnAfterLoad(IScene scene)
{
  // Add your code here, but make sure it is not a long process
}
```

AfterPlay

This event is triggered after the scene is taken to air. It will not be triggered if the scene is directly loaded into the preview channel. This event is triggered before the scene is loaded into preview.

Example:

Attach your event handler method **OnAfterPlay** to the **AfterPlay** event in the constructor of the scenes script class. Any time the **AfterPlay** event is triggered the **OnAfterPlay** method will be called.

```
// constructor
public ScriptTest()
{
      InitializeComponent();

      // Attach event handler to event
      this.AfterPlay += OnAfterPlay;

}


      …


// Handles the AfterPlay event

private void OnAfterPlay(IScene scene)
{
  // Add your code here, but make sure it is not a long process
}
```

BeforeStop

This event is triggered before the scene is transferred from air to preview. Clearing the scene from preview or air does not trigger this event. This event is triggered before the scene is loaded into preview.

Example:

Attach your event handler method **OnBeforeStop** to the **BeforeStop** event in the constructor of the scenes script class. Any time the **BeforeStop** event is triggered the **OnBeforeStop** method will be called.

```
// constructor
public ScriptTest()
{
      InitializeComponent();

      // Attach event handler to event
      this.BeforeStop -= OnBeforeStop;

}


      …


// Handles the BeforeStop event

private void OnBeforeStop(IScene scene)
{
  // Add your code here, but make sure it is not a long process
}
```

AfterStop

This event is triggered after the scene is transferred from air to preview. Clearing the scene from preview or air does not trigger this event. This event is triggered before the scene is loaded into preview.

Example:

Attach your event handler method **OnAfterStop** to the **AfterStop** event in the constructor of the scenes script class. Any time the **AfterStop** event is triggered the **OnAfterStop** method will be called.

```
// constructor
public ScriptTest()
{
      InitializeComponent();

      // Attach event handler to event
      this.AfterStop += OnAfterStop;

}


      …

// Handles the AfterStop event
private void OnAfterStop(IScene scene)
{
  // Add your code here, but make sure it is not a long process
}
```

Methods

Invoke

This method executes the specified delegate on the thread that owns the control's underlying window handle.

Syntax:

```
object Invoke(Delegate method)
```

Example:

To enable a **MyButton** button on the control panel the following code is needed if this is being done on a separate thread.

```
// Enable/Disable MyButton on control panel
private void EnableButton(bool enable)
{
      if ( this.InvokeRequired )
      {
            Invoke(new Action<bool>(EnableButton), new object [] { enable }
);
      }
      else
      {
            MyButton.Enabled = enable;
      }
}
```

FrameRate

Contains information about the frame rate.

Namespace: Chyron.Enterprise.Infrastructure.TimeCode

Properties

Name	Type	Description
FieldRate	double	Gets the field rate.
Interlaced	bool	Gets the flag indicating if the frame rate is interlaced.
Rate	double	Get the rate.
RoundedFieldRate	Int	Gets the FieldRate round to an interger.
RoundedRate	Int	Gets the Rate rounded to an interger.
Type	FrameRateType	The **FrameRateType** is an enumerated value that indicates the type of this frame rate. Below is a list of valid values and what they mean. *_24* 24 Hz *_25* 25 Hz *_30* 30 Hz *_29_97* 29.97 Hz *_48* 48 Hz *_50* 50 Hz *_60* 60 Hz *_59_94* 59.94 Hz *_23* 23 Hz *_100* 100 Hz

Resolution

Contains information about the resolution.

Namespace: **Chyron.PowerBox.Scene.Objects**

Properties

Name	Type	Description
Width	int	Gets the width in pixels for this resolution.
Height	bool	Gets the height in pixel for this resolution.
Size	Size	Gets the size (height and width) of this resolution.
FrameRate	FrameRate	Gets the frame rate for this resolution (see **FrameRate** section).
Interlaced	Bool	Gets a flag indicating that the resolution is interlaced,
PixelAspect	double	Gets the pixel aspect ratio for this resolution.
AspectRation	float	Gets the aspect ratio for this resolution.

Workflow Logger

Workflow logging is available to the scene scripts by using the runtime instance of the workflow manager object. There is a single method **LogMessage()** that can be used for logging to the **WorkFlow Monitor**.

Namespace: **Chyron.Framework.Application.WorkFlow**

Syntax:

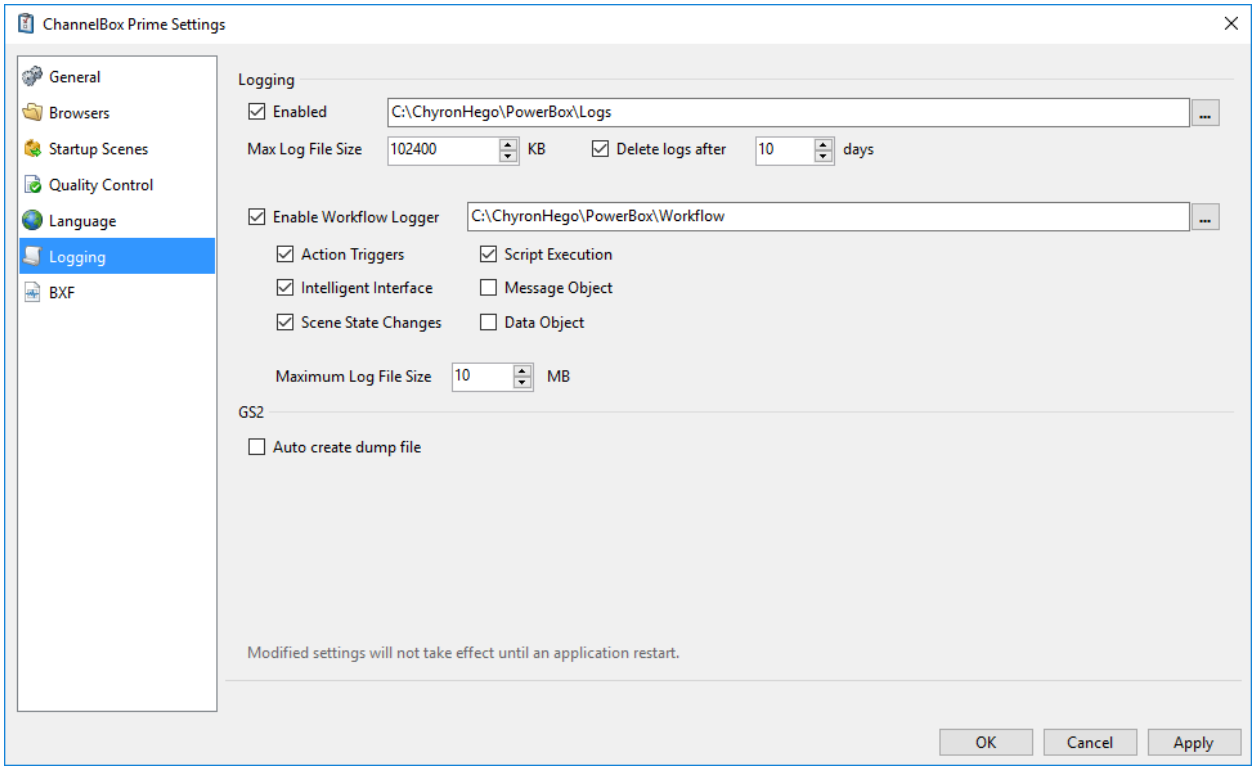
```
void LogMessage(WorkflowEvent eventType, string sceneName, string message)
```

Example:

```
using Chyron.Framework.Application.WorkFlow;

            ...

WorkflowManager.Instance.LogMessage(
                        WorkflowEvent.Message,
                        this.Scene.Name,
                        "My Message");

            ...
```

Workflow Configuration Settings

In order for logging to work, the user will need to enable it in the **PRIME** configuration settings form.



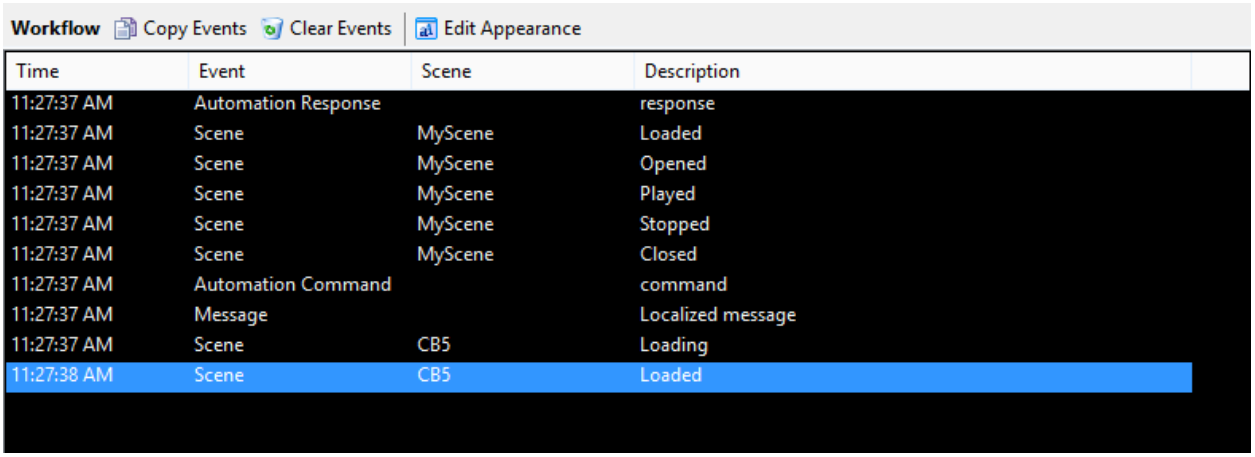
Logging Options

These options allow you to fine tune the events/messages that you want **PRIME** to log. These options will only be available if the Workflow Logger is enabled.

Action Triggers	If checked enables logging of action triggers.
Intelligent Interface	If checked enables logging of intelligent interface events.
Scene State Changes	If checked enables logging of scene state changes.
Script Execution	If checked allows logging of script execution.

Workflow Monitor

The Workflow monitor available through the **PRIME** application will display the log entries sent using the Workflow logger methods list above.



Control Panel Objects

Each scene has a set of controls that are collectively referred to as the control panel. These controls are exposed to scripting, providing specialized properties and functions pertinent to each control type. From the scenes C# script controls are accessed using just their name. For example, adding a new button control to the scenes control panel will automatically be named **"Button 1".** Accessing this control from the scenes C# script will be by the name **Button1**. Although the name of a control can contain spaces these spaces are removed in the scripting objects name. Accessing a scenes control through the Application Script is different and will be described in another section.

All controls implement the ISceneObject interface, so all properties, events and methods from this object are available to all of these objects. Most control panel controls correspond to controls in the Microsoft C# library. To get a more complete list of properties, events and methods for each of the controls in this section, refer to Microsoft's documentation.

Note – The **FilePicker**, **Time Code Editor** and **Asset Browser** do not correspond to a Microsoft C# control.

Button

The Button can be used to interact with the scene by performing a process when it is clicked.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Text	string	Gets or sets the buttons label text.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

Click

Event triggered when the button is clicked with the mouse.

Syntax:

```
void Click(object sender, EventArgs args)
```

Example: This example registers a Click event handler for MyButton Button control. The handler will change the Buttons *Text* property to “Stop” if the current text is “Play” and stop the scenes “MyAction” action. Otherwise if the Button’s text is not “Play” it will change the Buttons *Text* property to “Stop” and play the scenes “MyAction” action.

```
public ScriptTest()

{
       InitializeComponent();


       MyButton.Click += OnButtonClicked;

}


// Handles the Click event from MyButton
public void OnButtonClicked(object sender, EventArgs args)
{

   var action = this.Scene.Actions.Find("MyAction");


    if ( MyButton.Text == "Play" )
    {
       MyButton.Text = "Stop";
       action.Play();
    }
    else
    {
       MyButton.Text == "Play"
       action.Stop();
    }
}
```

Label

The **Label** can be used to help identify fields on the control panel by setting the Text property.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Text	string	Gets or sets the labels text.

TextBox

The **TextBox** can be used to interact with a scene through binding to graphic controls or to display context-based text generated within script.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Text	string	Gets or sets the text that appears in the text box.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

TextChanged

This event is triggered when the **TextBox** controls *Text* property has changed. Note that this event us triggered for every character entered or removed. Register a handler if you want to perform an operation when the *Text* property is changed in the **TextBox**.

Syntax:

```
void TextChanged(object sender, EventArgs args)
```

Example

```
public ScriptTest ()
{
    InitializeComponent();

    // Get the SAMPLE scene from the ChannelBox database
    var scene = ChannelBox.GetScene("SAMPLE");

    // Hook up the event handler for the PlayoutStateChange event
    scene.PlayoutStateChanged += OnPlayoutStateChanged;

}

// Handles the PlayoutStateChanged event from the scene.
private void OnPlayoutStateChanged(IChannelElement element)
{
    // Set the TextBox text to the curent playout state
    SceneStateTextBox.Text = element.PlayoutState.ToString();
}
```

ComboBox

The **ComboBox** can be used to interact with a control panel combo box. Below is a list of some of the **ComboBox** properties and events of interest. To get a more complete list refer to Microsoft's documentation.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
SelectedItem	Item Type	Gets the current selected item in the combo box or null if no items are selected. Returns an object which is the type of the items in the ComboBox.
Items	ObjectCollection	Gets the collection of items for the ComboBox. This property allows the addition and removal of items in the ComboBox.
Items.Count	int	Gets the number of items in the ComboBox.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

SelectedIndexChanged

The event is triggered when the selected item is changed in the **ComboBox**. Register a handler if you want to perform an operation when the *SelectedItem* property is changed in the **ComboBox**.

Example

Fill in the combo box with the list of all actions in the current scene. Register **OnSelectedIndexChanged** handler with the **SelectedIndexChanged** event from **MyComboBox**. The **OnSelectedIndexChanged()** handler will get the action name from the **ComboBox.SelectedItem** property and look up action in the current scenes action list. If the action is found (not equal to null) and the action is not already playing then execute the **Play()** method of the action.

```
public ScriptTest()
{
	InitializeComponent();


	foreach( vat action in this.Scene.Actions )
	{
		MyComboBox.Items.Add(action.Name);
	}

	MyComboBox.SelectedIndexChanged += OnSelectedIndexChanged;

}

// Handles the SelectedIndexChanged for MyComboBox
private void OnSlectedIndexChanged(object sender, EventArgs args)
{
	var name = MyComboBox.SelectedItem as string;

	var action = this.Scene.Actions.Find(name);

	if ( action !=- null && !action.Playing)
	{
		action.Play();
	}
}
```

CheckBox

The **CheckBox** can be used to interact with the scene by performing a process when it is checked or unchecked.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Checked	bool	Gets the flag indicating if the checkbox in checked (true) or not (false).

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

CheckedChanged

The event is triggered when the check state of the CheckBox has changed.

Syntax:

```
void CheckedChanged(object sender, EventArgs args)
```

Example:

```
public ScriptTest()
{
	InitializeComponent();


	MyCheckBox.CheckedChanged += OnCheckedChanged;


}

// Handles the CheckedChanged event from MyCheckBox
public void OnCheckedChanged(object sender, EventArgs args)
{
    // Your code for handling the checkbox checkedchanged event
}
```

RadioButton

The **RadioButton** can be used to interact with the scene by performing a process when it is checked or unchecked.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Checked	bool	Gets the flag indicating if the radio button in checked (true) or not (false).

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

CheckedChanged

The event is triggered when the check state of the RadioButton has changed.

Syntax:

```
void CheckedChanged(object sender, EventArgs args)
```

Example:

```
public ScriptTest()

{
      InitializeComponent();

      MyRadioButton.CheckedChanged += OnCheckedChanged;



}


// Handles the CheckedChanged event from MyRadioButton
public void OnCheckedChanged(object sender, EventArgs args)
{
    // Your code for handling the radio button checkedchanged event
}
```

NumericUpDown

The **NumericUpDown** can be used to interact with the current scene by setting a numeric value.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Value	decimal	Gets the value contained in the control.
Minimum	decimal	Gets or sets the minimum value that the **NumericUpDown** Value can be set.
Maximum	decimal	Gets or sets the maximum value that the **NumericUpDown** Value can be set.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

ValueChanged
This event is triggered when the **NumericUpDowns** controls *Value* property has changed.

Syntax:

```
void ValueChanged(object sender, EventArgs args)
```

Example:

```
public ScriptTest ()
{
    InitializeComponent();

    // Register handler
    MyNumericUpDown.ValueChanged += OnValueChanged;
}


// Handles the ValueChanged event from the NumbericUpDown control
public void OnValueChanged(object sender, EventArgs args))
{
    // Your code to handle the ValeuChanged event of the NumericUpDown control
}
```

TrackBar

The **TrackBar** can be used to interact with the current scene by setting a numeric value.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Value	decimal	Gets or sets the value contained in the control.
Minimum	decimal	Gets or sets the minimum value that the **TrackBar** Value can be set.
Maximum	decimal	Gets or sets the maximum value that the **TrackBar** Value can be set.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

ValueChanged
This event is triggered when the **TrackBars** controls *Value* property has changed.

Syntax:

```
void ValueChanged(object sender, EventArgs args)
```

Example:

```
public ScriptTest ()
{
    InitializeComponent();

    // Register handler
    MyTrackBar.ValueChanged += OnValueChanged;
}


// Handles the ValueChanged event from the TrackBar control
public void OnValueChanged(object sender, EventArgs args))
{
    // Your code to handle the ValeuChanged event of the TrackBar control
}
```

ListView

The **ListView** can be used to display lists of items in the control panel. This object is mostly used by binding it to a **DataObject** to display its contents.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
SelectedItems	SelectedListViewItemCollection	Gets the list of selected ListViewItems in the ListView object.
SelectedIndices	SelectedIndexCollection	Gets the list of selected ListViewItems by index in the ListView.Items collection.
Items	ListViewItemCollection	Gets the collection of ListViewItems for the ListView. This property allows the addition and removal of items in the ListView.
Items.Count	int	Gets the number of items in the ListView.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

SelectedIndexChanged

The event is triggered when the selected item is changed in the **ListView**. Register a handler if you want to perform an operation when the *SelectedItems* property is changed in the **ComboBox**.

Example:

Register **OnSelectedIndexChanged** handler with the **SelectedIndexChanged** event from **MyListView**. The **OnSelectedIndexChanged()** handler will get the

```
public ScriptTest()
{
	InitializeComponent();


	MyListView.SelectedIndexChanged += OnSelectedIndexChanged;

}

// Handles the SelectedIndexChanged for MyListView
private void OnSlectedIndexChanged(object sender, EventArgs args)
{
	foreach ( var item in MyListView.Items.OfType<ListViewItem>() )
	{
		// Your code to process selected ListViewItems
	}
}
```

FilePicker

The **FilePicker** can be used to interact with the scene by allowing files to be selected so that File properties on other objects can be modified.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
File	string	Gets the file path selected by the file picker.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

FileChanged

The event is triggered when the file is changed in the FilePicker control.

Syntax:

```
void FileChanged()
```

Example

Register **OnFileChanged** handler with the **FileChanged** event from **MyFilePicker** so that some processing can be performed in the **OnFileChanged()** handler.

```
public ScriptTest()
{
      InitializeComponent();

      MyFilePicker.FileChanged += OnFileChanged;

}

// Handles the FileChanged for MyFilePicker
private void OnFileChanged()
{
      var file = MyFilePicker.File;

      // You code to handle the file changed event from the FilePicker

}
```

AssetBrowser

The **AssetBrowser** can be used to interact with the scene by allowing asset files to be selected for the current projects.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
File	string	Gets the file path selected by the asset browser.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

FileChanged

The event is triggered when the file is selected in the **AssetBrowser** control.

Syntax:

```
void FileChanged()
```

Example:

Register **OnFileChanged** handler with the **FileChanged** event from **MyAssetBrowser** so that some processing can be performed in the **OnFileChanged()** handler.

```
public ScriptTest()
{
	InitializeComponent();

	MyAssetBrowser.FileChanged += OnFileChanged;

}

// Handles the FileChanged for MyAssetBrowser
private void OnFileChanged()
{
	var file = MyFilePicker.File;

	// You code to handle the file changed event from the AssetBrowser

}
```

PictureBox

The **PictureBox** can be used to interact with the scene by allowing images to be displayed in the control panel.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Image	Image	Gets or sets the image to be displayed in the picture box.

Example:

Sets the image of the picture box when the image file is selected from the system.

```
public ScriptTest()
{
      InitializeComponent();

      ImageFilePicker.FileChanged += OnFileChanged;

}

// Handles the FileChanged for ImageFilePicker
private void OnFileChanged()
{
      var file = ImageFilePicker.File;

      MyPictureBox.Image = new Bitmap(file);

 }
```

TimeCodeEditor

The **TimeCodeEditor** can be used to interact by using a timer to trigger actions or processes.

Properties

Name	Type	Description
Name	string	Gets the name of the control
Enabled	bool	Gets the flag indicating if the control is enabled (true) or disabled (false).
Visible	bool	Gets the flag indicating if the control is visible (true) or not (false).
Frames	int	Gets or sets the number of frames.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

FramesChanged

This event is triggered when the **TimeCodeEditor** controls frames have changed.

Syntax:

```
void FranmesChanged()
```

Example:

```
public ScriptTest()
{
      InitializeComponent();

      MyTimeCodeEditor.FramesChanged += OnFramesChanged;

}

// Handles the FramesChanged event for MyTimeCodeEditor
private void OnFramesChanged()
{
      // You code to handle the FramesChanged event
}
```

Canvas Objects

These controls are graphics based controls that implement the **IDynamicObject** and **ISceneObject** interfaces, so all properties, event and methods from these interfaces are implemented in the following objects.

Text

Object used for 2D and 3D text display.

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Text	string	Gets or sets the text for this graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.
_2d	_2dTextProperties	Gets the 2D text properties.
_3d	_3dTextProperties	Gets the 3D text properties.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

TextChanged

Event is triggered when the Text is property is changed in the Text graphics object.

Syntax:

```
void TextChanged(ISceneObject sceneObject)
```

Example:

Image

Object used for displaying images on the Canvas.

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.
File	string	Gets or sets the image file to be displayed within this graphics object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

ImageChanged

Event is triggered when the Image is property is changed in the Image graphics object.

Syntax:

```
void ImageChanged(ISceneObject sceneObject)
```

Example:

Clip

Object used for clips in the Canvas.

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.
File	string	Gets or sets the image file to be displayed within this graphics object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Finished

Event is triggered when the clip finishes playing.

Syntax:

```
void Finished(ISceneObject sceneObject)
```

Example:

Cube

Object used for 3D cube object in the Canvas.

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.
File	string	Gets or sets the image file to be displayed within this graphics object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Model

Object used for model on the Canvas.

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.
File	string	Gets or sets the image file to be displayed within this graphics object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

FileChanged

Event is triggered when the File property of the Model is changed.

Syntax:

```
void FileChanged(ISceneObject sceneObject)
```

Example:

Effects

These controls are graphics effects that may be associated with graphical controls. These objects implement the **IDynamicObject** and **ISceneObject** interfaces, so all properties, events and methods from these interfaces are implemented in the following objets.

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Crop

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Mask

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Material

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Light

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Render Texture

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Warp

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Page Turn

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Transition

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Crawl

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Example:

The scene uses a FilePicker control on the Control Panel to allow the file used in the crawl effect to be changed while the scene is on air. It handles the FileChanged event of the FilePicker control to stop the current crawl update the file and restart.

```
public ScriptTest()
{
  InitializeComponent();

}


private void MyFilePicker_FileChanged()
{
   var filePath = MyFilePicker.File;

   if ( ! string.IsNullOrWhiteSpace(filePath) &&
System.IO.File.Exists(filePath) )
   {
      MyCrawl.Stop();
      MyCrawl.File = filePath;
      MyCrawl.Start();
   }
}
```

Character

Properties

Name	Type	Description
Name	string	Gets the name of the graphic object.
Enabled	bool	Gets the flag indicating if the graphics object is enabled (true) or disabled (false).
Scene	IScene	Gets the scene this object belongs to.
Opacity	double	Gets or sets the opacity or the graphic object.
Effects	SceneObjectList<IEffect>	Gets the list of effects associated with this graphics object.
Animations	AnimationList	Gets the list of animations associated with this graphic object.
Events	EventList	Gets the list of events associated with this graphic object.

Events

BeforePropertyChanged

See BeforePropertyChanged in ISceneObject Events section.

PropertyChanged

See PropertyChanged in ISceneObject Events section.

AfterPropertyChanged

See AfterPropertyChanged in ISceneObject Events section.

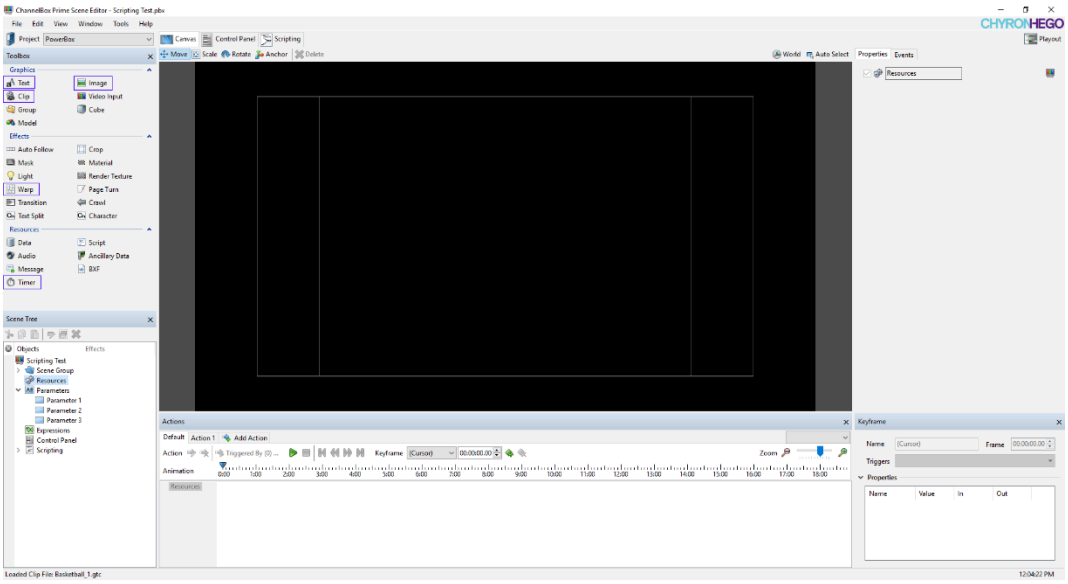
PropertyAnimated

See PropertyAnimated in IDynamicObject Events section.

Usage

Canvas Objects

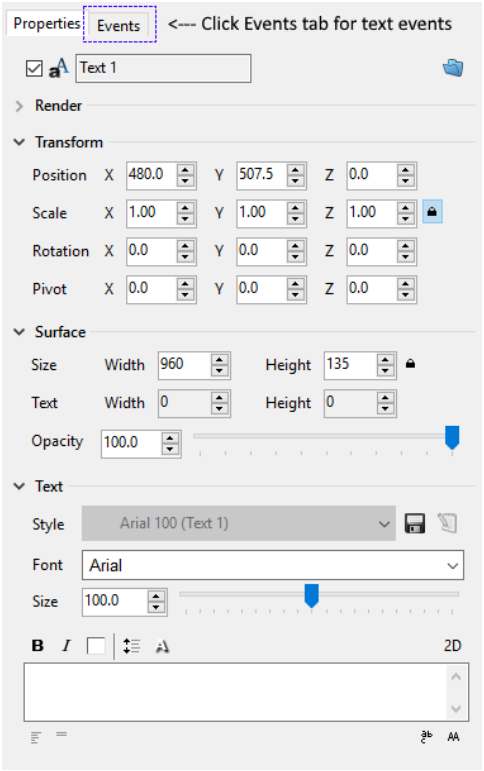
Many canvas objects have events that can trigger C# scripting code. The figure below shows an image of the Canvas with the toolbox objects that have events available for C# scripting highlighted.



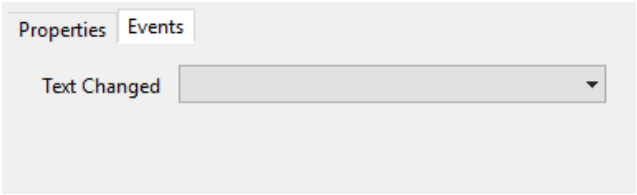
Graphics

Text

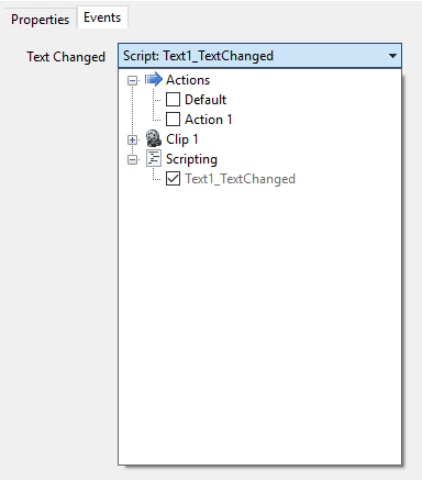
An author can add a text object to the scene by clicking the Text button Text of the Toolbox panel on the left hand side of the canvas. Once added you will see the text properties on the right hand side of the canvas.



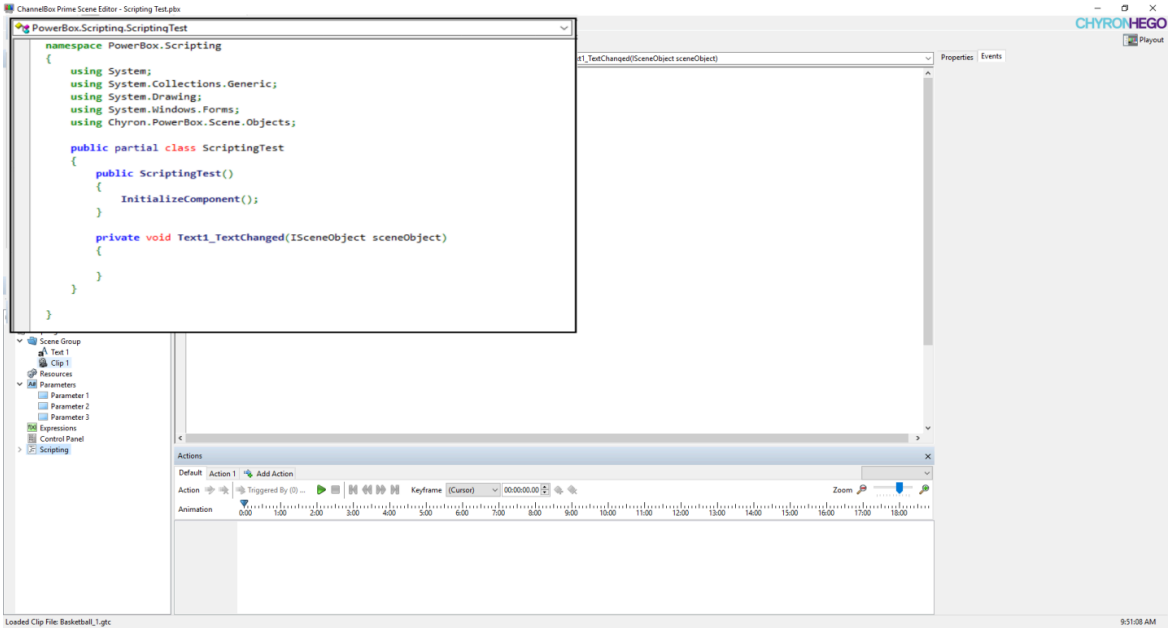
Click on the **Events** tab to view the event triggers available to the text object as seen below.



Clicking on the **Text Changed** drop down box will display the list of items that can be triggered when the text is changed.



Check the Text1_TextChanged item under **Scripting** and the **Text1_TextChanged()** method will be automatically added to your scene script.

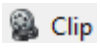


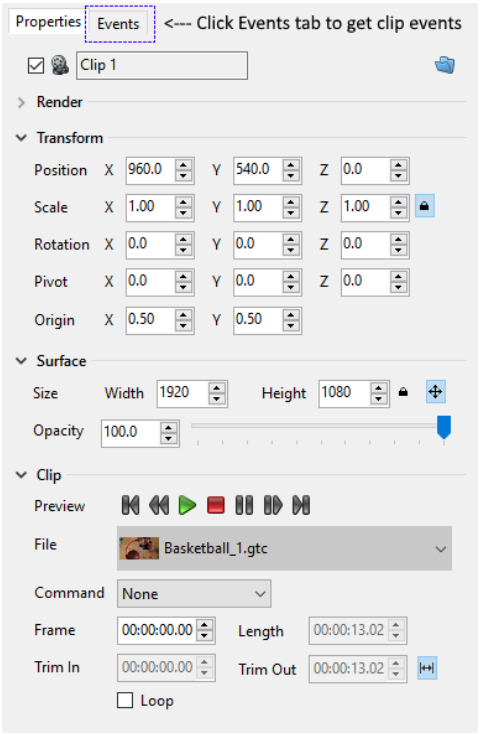
From the script editor the author may enter the code to be executed when the text objects **TextChanged** event is triggered see the sample below.

Sample Usage:

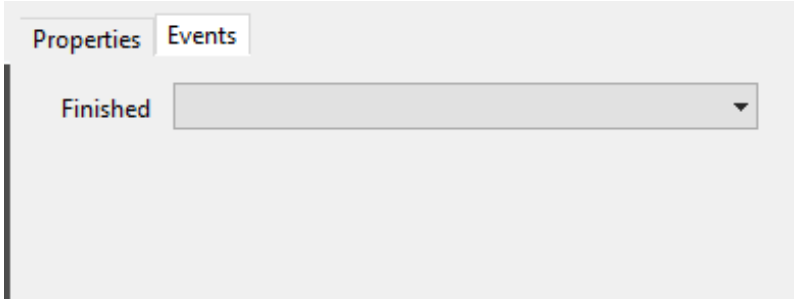
```
private void Text1_TextChanged(ISceneObject sceneObject)
{
    // Enter your code here
}
```

Clip

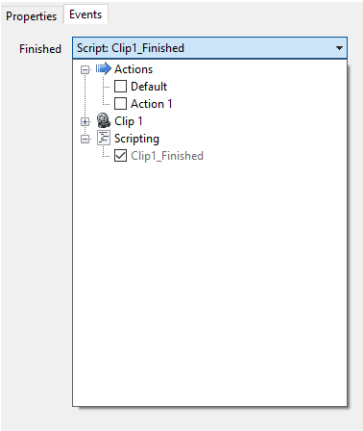
An author can add a clip to the scene by clicking the Clip button  of the Toolbox panel on the left hand side of the canvas. Once added you will see the clip properties on the right hand side of the canvas.



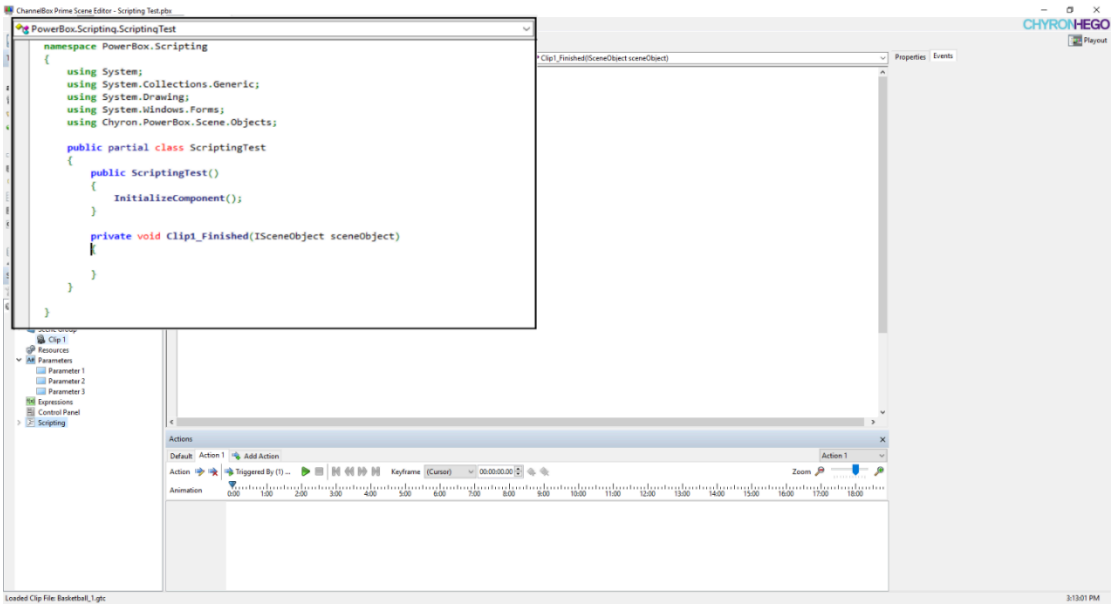
Click on the Events tab to view the events triggers available to the clip as below.



Clicking on the Finished drop down box will display the list of items that can be triggered when the clip finishes.



Check the Clip1_Finished item under **Scripting** and the **Clip1_Finished()** method will be automatically added to your scene script.



From the script editor the author may enter the code to be executed when the clip **Finished** event is triggered see the sample below.

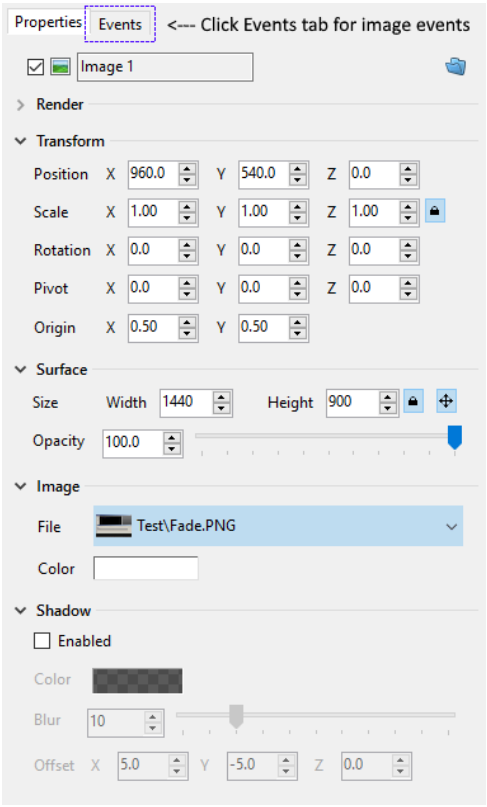
Sample Usage:

```
private void Clip1_Finished(ISceneObject sceneObject)
{
    // Enter your code here
}
```

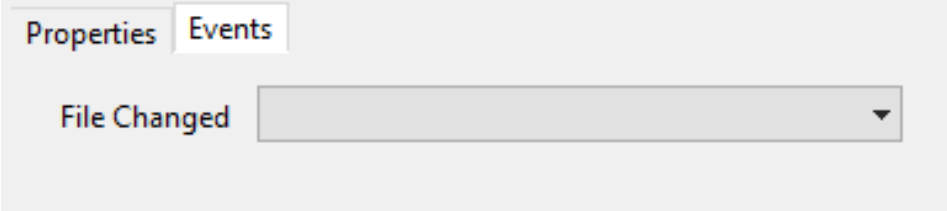
Image

An author can add a clip to the scene by clicking the Image button 

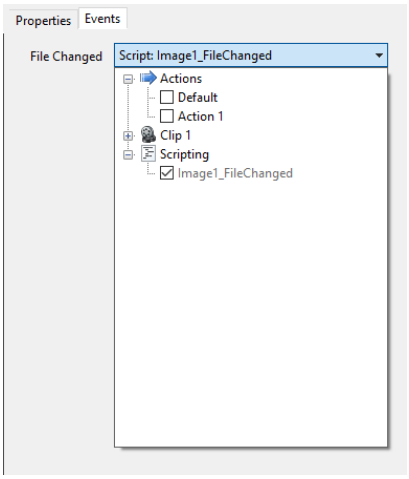
of the Toolbox panel on the left hand side of the canvas. Once added you will see the image properties on the right hand side of the canvas.



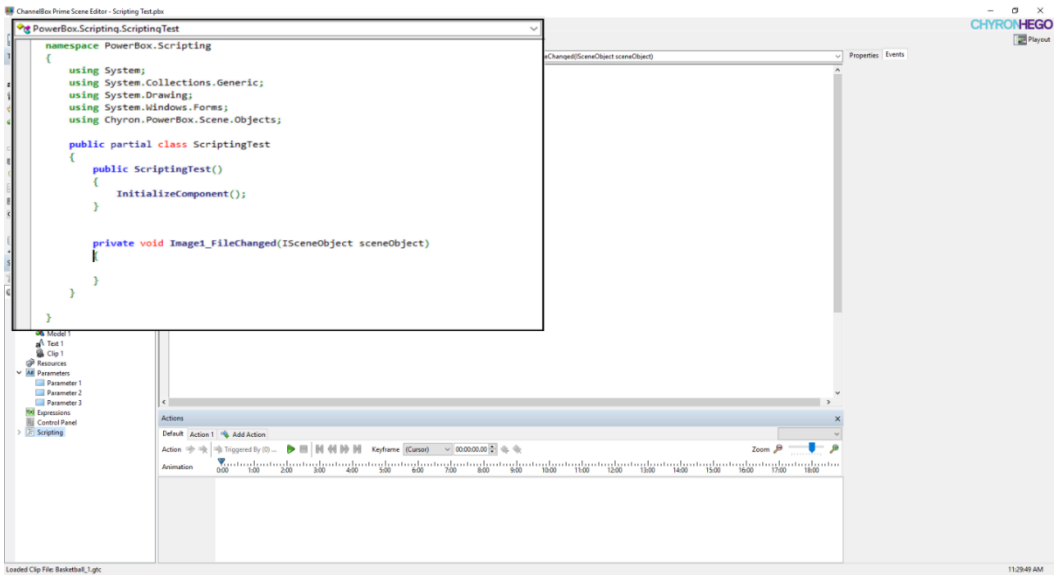
Click on the Events tab to view the event triggers available to the image object as seen below.



Clicking on the File Changed drop down box will display the list of items that can be triggered when the images file is changed.



Check the Image1_FileChanged item under **Scripting** and the **Image1_FileChanged()** method will be automatically added to your scene script.



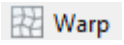
From the script editor the author may enter the code to be executed when the images **FileChanged** event is triggered see the sample below.

Sample Usage:

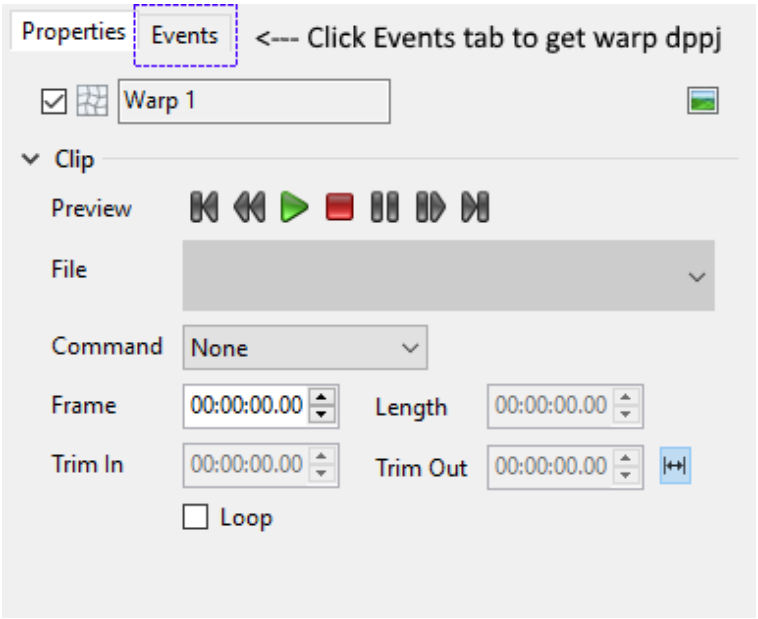
```
private void Image1_FileChanged(ISceneObject sceneObject)
{
    // Enter your code here
}
```

Effects

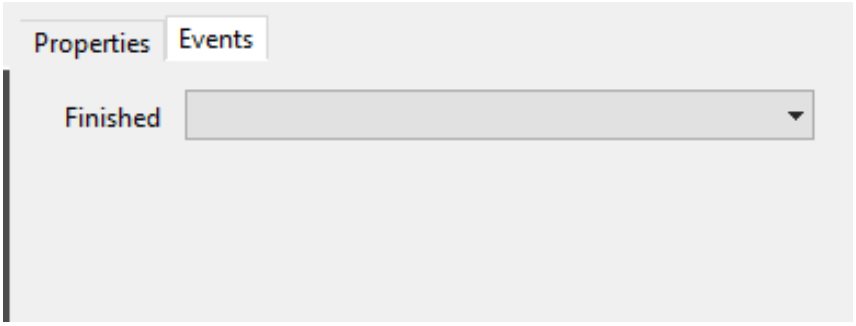
Warp

An author can add a warp effect to a scene object by clicking the Warp button 

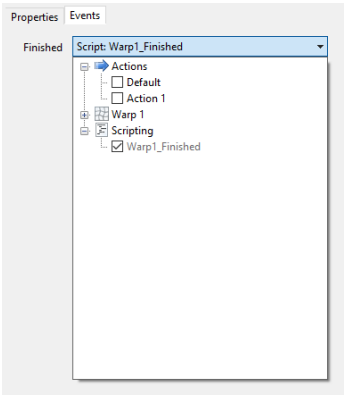
of the Toolbox panel on the left hand side of the canvas. Once added you will see the warp properties on the right hand side of the canvas.



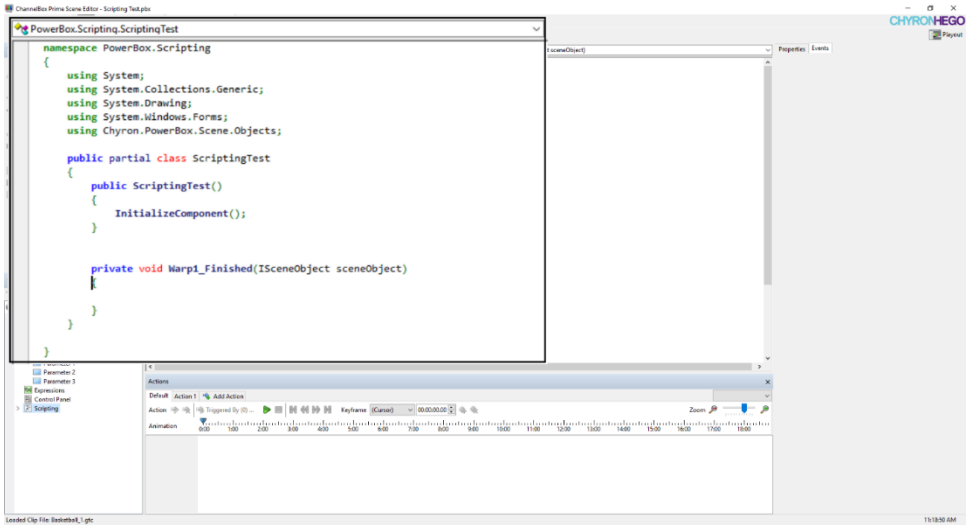
Click on the Events tab to view the events triggers available to the warp effect as seen below.



Clicking on the Finished drop down box will display the list of items that can be triggered when the warp effect finishes.



Check the Warp1_Finished item under **Scripting** and the **Warp1_Finished()** method will be automatically added to your scene script.

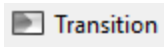


From the script editor the author may enter the code to be executed when the warp effects **Finished** event is triggered see the sample below.

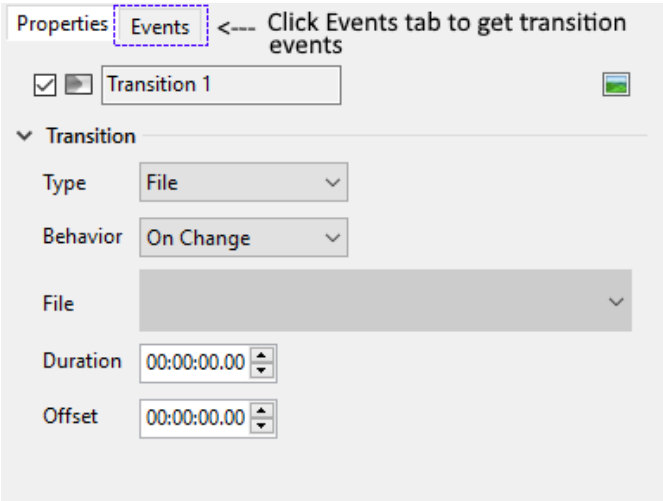
Sample Usage:

```
private void Warp1_Finished(ISceneObject sceneObject)
{
    // Enter your code here
}
```

Transition

An author can add a transition effect to a scene object by clicking the Transition button 

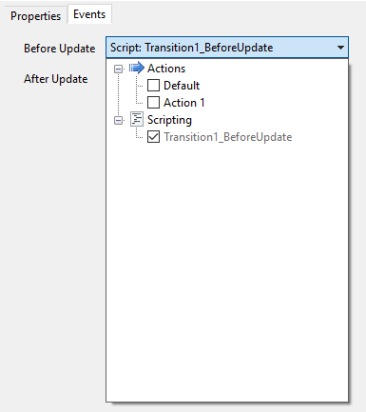
of the Toolbox panel on the left hand side of the canvas. Once added you will see the warp properties on the right hand side of the canvas.



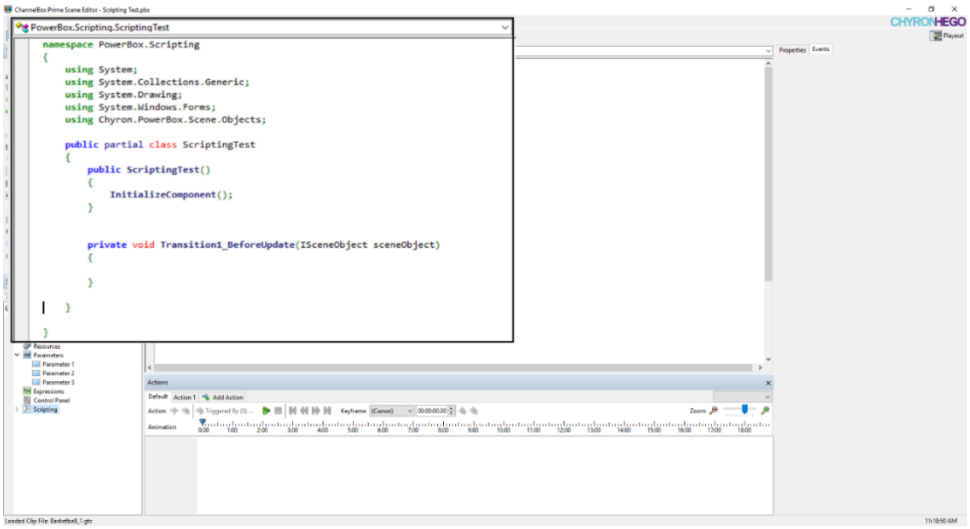
Click on the Events tab to view the events triggers available to the transition effect as seen below.

Transition before Update Events

Clicking on the Before Update drop down box will display the list of items that can be triggered before the transition effect updates the associated scene object.



Check the Transition1_BeforeUpdate item under **Scripting** and the **Transition1_BeforeUpdate()** method will be automatically added to your scene script.



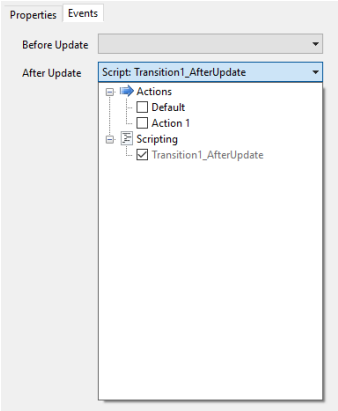
From the script editor the author may enter the code to be executed when the transition effects **BeforeUpdate** event is triggered see the sample below.

Sample Usage:

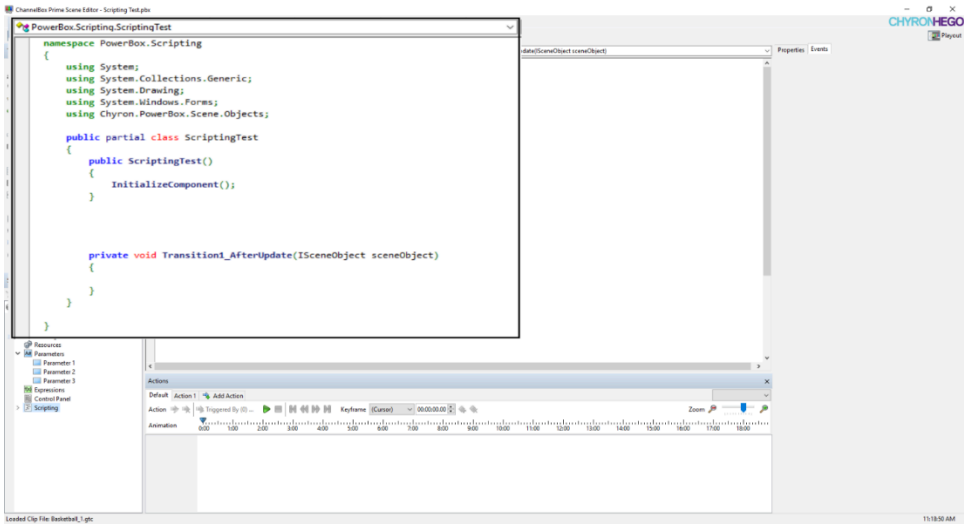
```
private void Transition1_BeforeUpdate(ISceneObject sceneObject)
{
    // Enter your code here
}
```

Transition after Update Events

Clicking on the After Update drop down box will display the list of items that can be triggered before the transition effect updates the associated scene object.



Check the Transition1_AfterUpdate item under **Scripting** and the **Transition1_AfterUpdate()** method will be automatically added to your scene script.



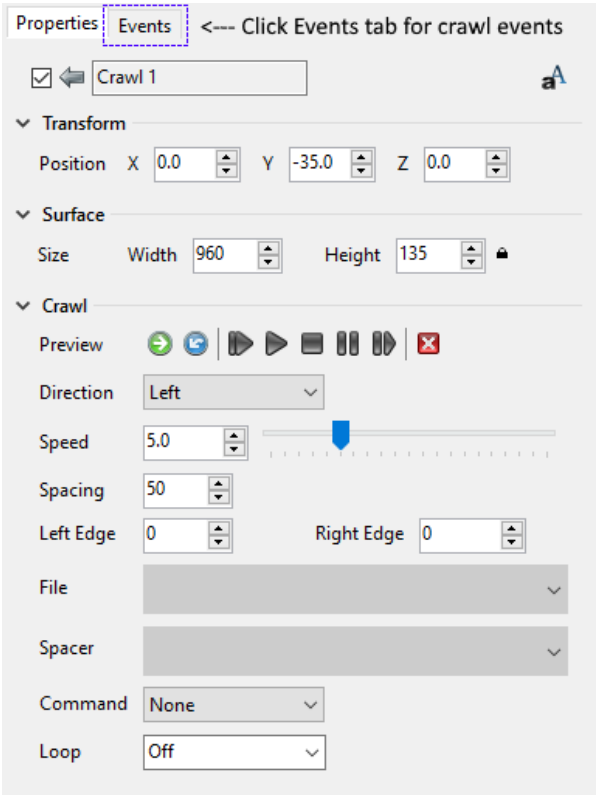
From the script editor the author may enter the code to be executed when the transition effects **AfterUpdate** event is triggered see the sample below.

Sample Usage:

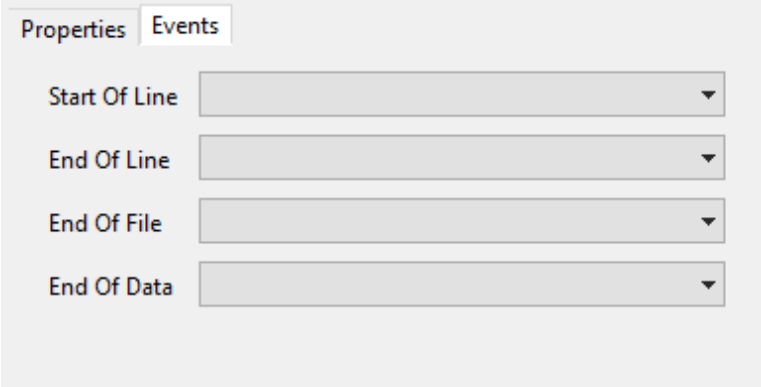
```
private void Transition1_AfterUpdate(ISceneObject sceneObject)
{
    // Enter your code here
}
```

Crawl

An author can add a crawl effect to a scene object by clicking the Warp button Crawl of the Toolbox panel on the left hand side of the canvas. Once added you will see the crawl properties on the right hand side of the canvas.

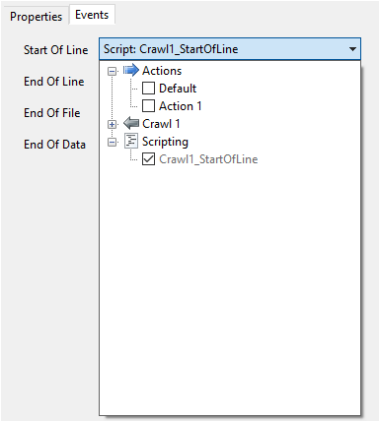


Click on the Events tab to view the events triggers available to the warp effect as seen below.

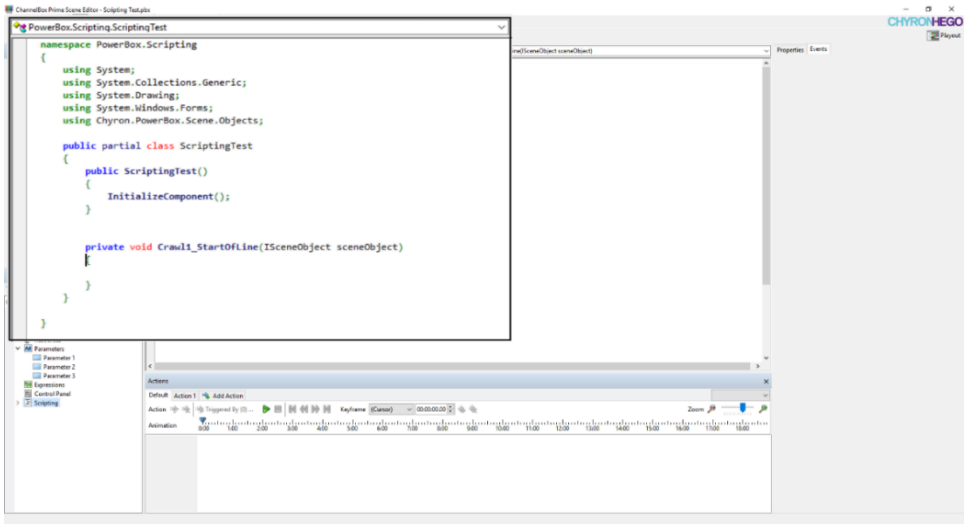


Start of Line Event

Clicking on the Start Of Line drop down box will display the list of items that can be triggered when the crawl effect is starting a new line.



Check the Crawl1_StartOfLine item under **Scripting** and the **Crawl1_StartOfLine()** method will be automatically added to your scene script.



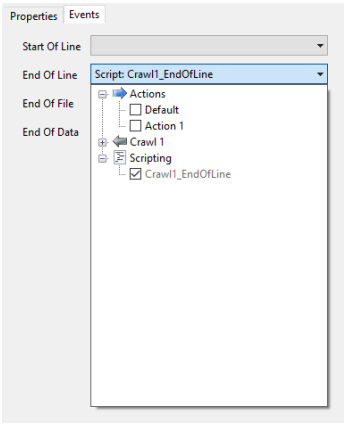
From the script editor the author may enter the code to be executed when the crawl effects **StartOfLine** event is triggered see the sample below.

Sample Usage:

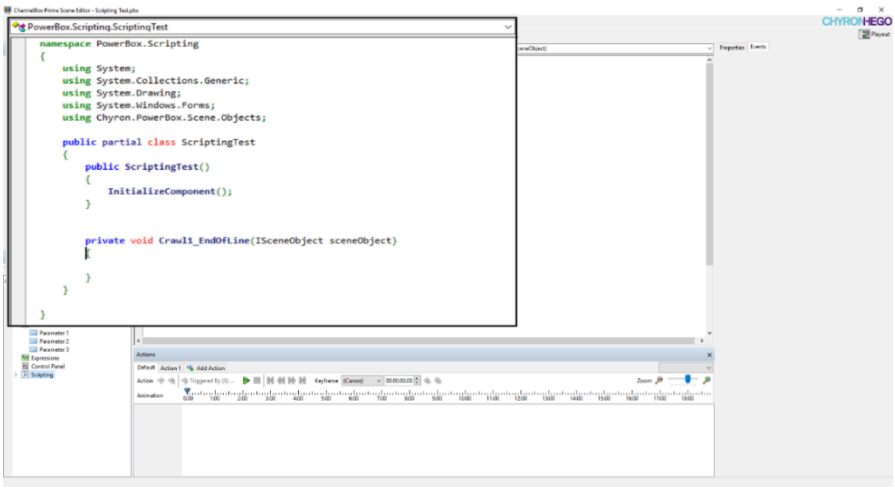
```
private void Crawl1_StartOfLine(ISceneObject sceneObject)
{
    // Enter your code here
}
```

End of Line Event

Clicking on the End Of Line drop down box will display the list of items that can be triggered when the crawl effect is ending a line.



Check the Crawl1_EndOfLine item under **Scripting** and the **Crawl1_EndOfLine()** method will be automatically added to your scene script.



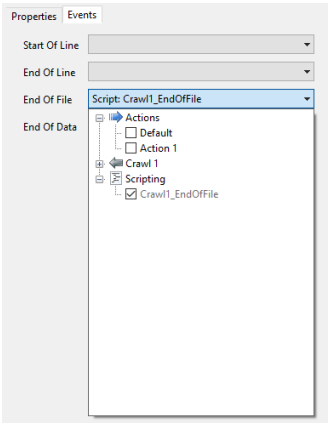
From the script editor the author may enter the code to be executed when the crawl effects **EndOfLine** event is triggered see the sample below.

Sample Usage:

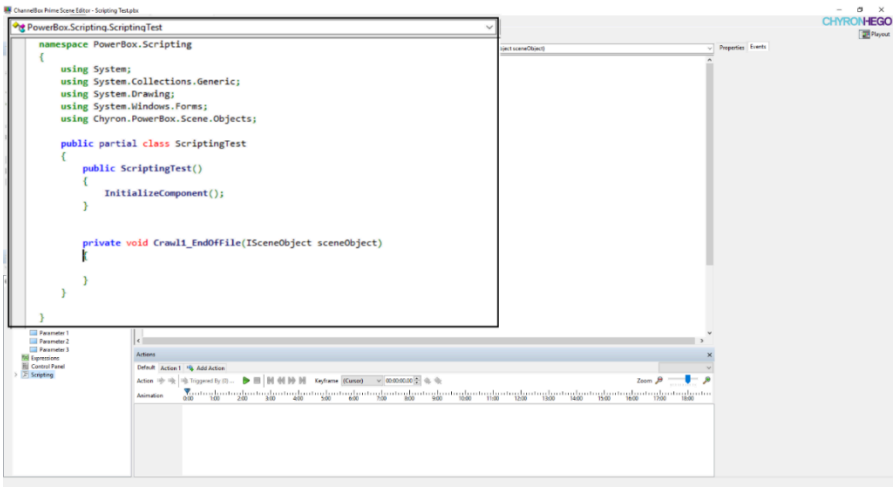
```
private void Crawl1_EndOfLine(ISceneObject sceneObject)
{
    // Enter your code here
}
```

End of File Event

Clicking on the End Of File drop down box will display the list of items that can be triggered when the crawl effect reaches the end of file.



Check the Crawl1_EndOfFile item under **Scripting** and the **Crawl1_EndOfFile()** method will be automatically added to your scene script.



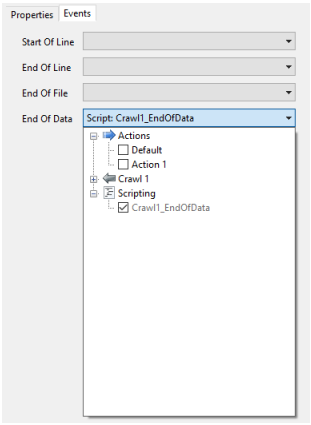
From the script editor the author may enter the code to be executed when the crawl effects **EndOfFile** event is triggered see the sample below.

Sample Usage:

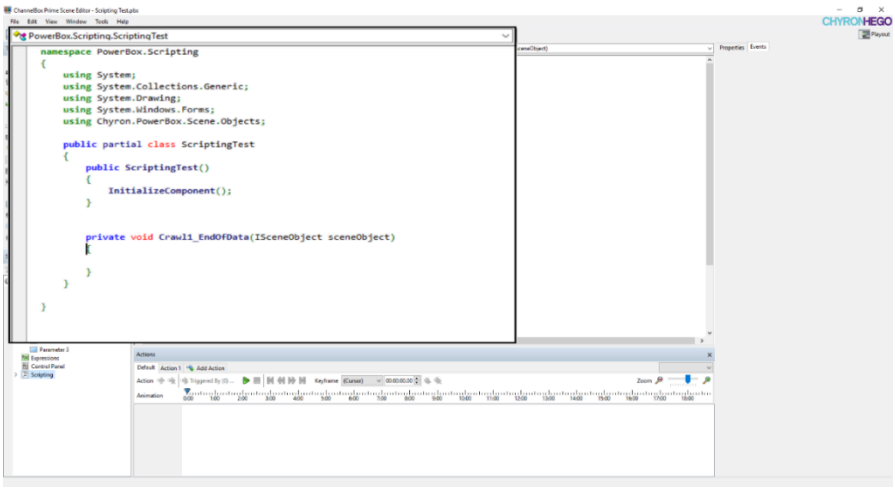
```
private void Crawl1_EndOfFile(ISceneObject sceneObject)
{
    // Enter your code here
}
```

End of Data Event

Clicking on the End Of Data drop down box will display the list of items that can be triggered when the crawl effect is at the end of data.



Check the Crawl1_EndOfData item under **Scripting** and the **Crawl1_EndOfData()** method will be automatically added to your scene script.



From the script editor the author may enter the code to be executed when the crawl effects **EndOfData** event is triggered see the sample below.

Sample Usage:

```
private void Crawl1_EndOfData(ISceneObject sceneObject)
{
    // Enter your code here
}
```

Resources

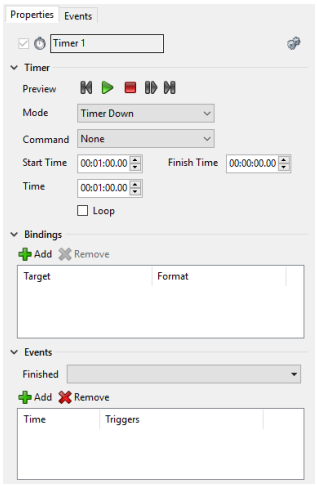
Timer

Timer Properties

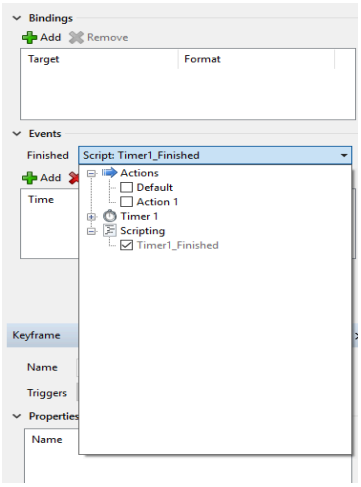
Finished Event

A scene author can add a timer object to the scene by clicking the Timer button 

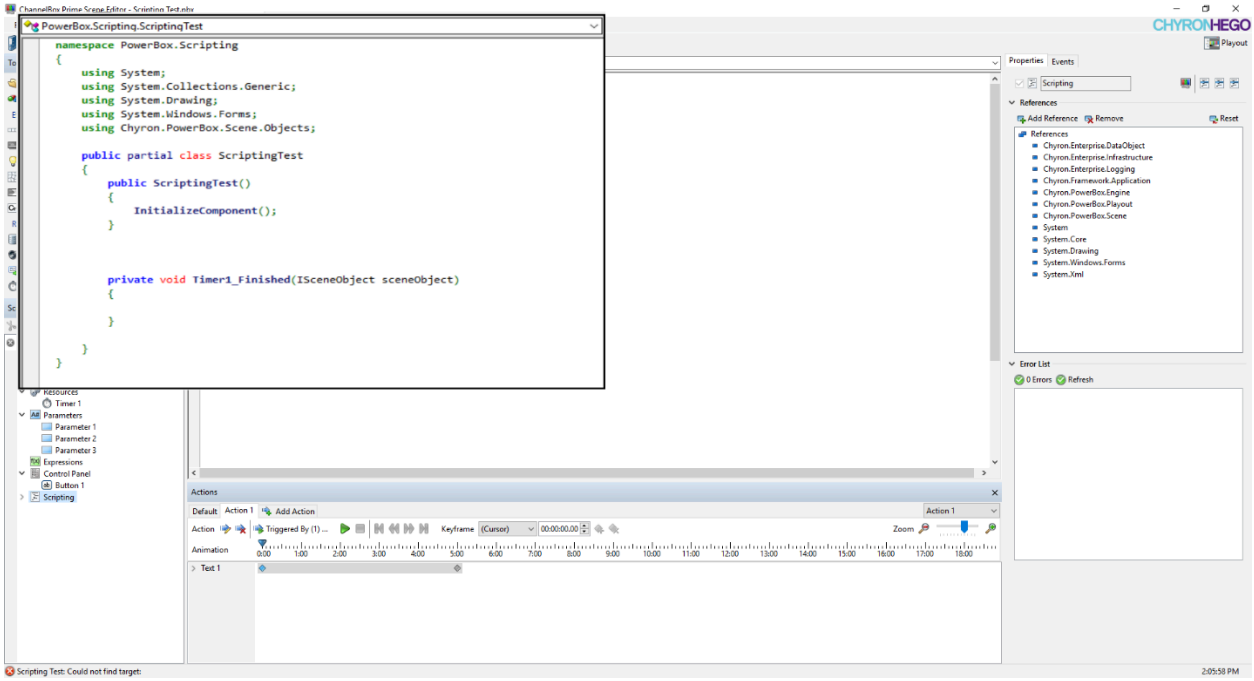
in the Toolbox panel on the left hand side. Once added they will see the timer properties on the right hand side of the scene canvas.



The author can then add a Finished event through the properties tab that gets triggered when the timer ends.



Clicking on the Finished drop down box a list of items that can be triggered is presented. Check the Timer1_Finished item under **Scripting** and the **Timer1_Finished()** method will be automatically added to your scene script.



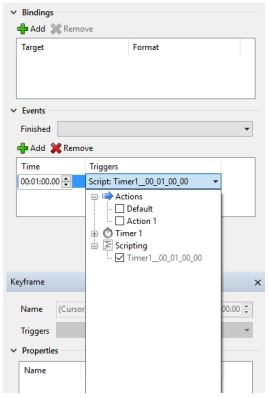
From the script editor the author may enter the code to be executed when the timed event is triggered see the sample below.

Sample Usage:

```
private void Timer1_Finished(ISceneObject sceneObject)
{
    // Your code here
}
```

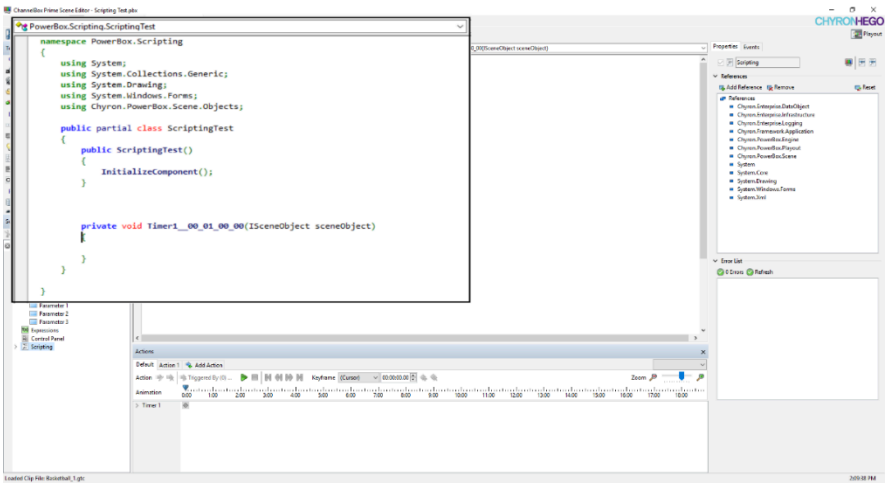
Timed Event

A scene author can add a timed event to be triggered from the timer properties.



Clicking the 

button in the Timer Events property section will add an event that can be triggered based on the time set in the *Timer* column. Click the Triggers next to the Time and a list of available items that can be triggered will be displayed. Checking the Timer1_00_01_00_00 will add the **Timer1_00_01_00_00()** method automatically to the scenes script.



From the script editor the author may enter the code to be executed when the timed event is triggered see the sample below.

Sample Usage:

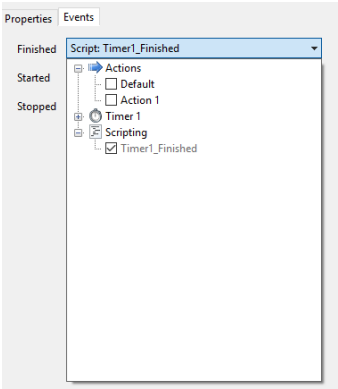
```
private void Timer1_00_01_00_00(ISceneObject sceneObject)
{
    // Your code here
```

```
}
```

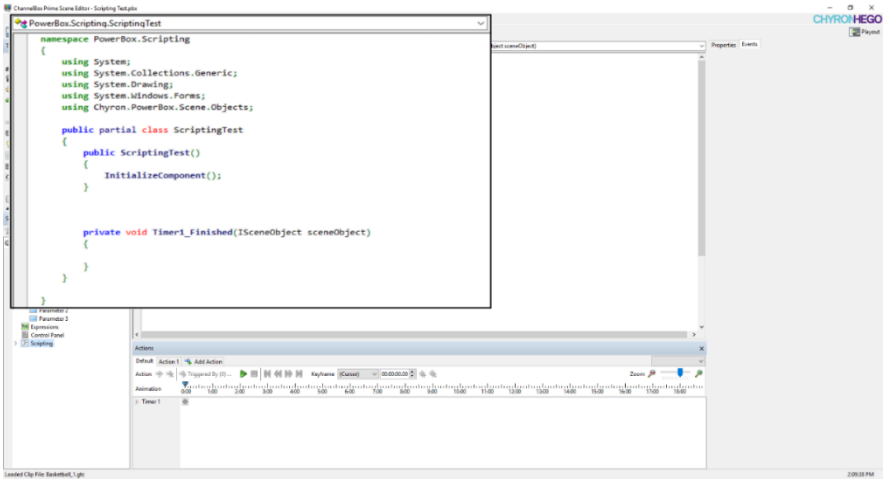
Timer Events

Finished Event

The scene author can add a Finished event through the events tab that gets triggered when the timer finishes.



Clicking on the Finished drop down box a list of items that can be triggered is presented. Check the Timer1_Finished item under Scripting and the **Timer1_Finished()** method will be automatically added to your scene script.



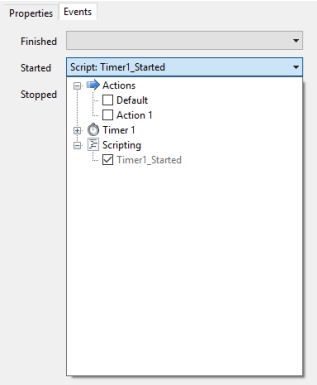
From the script editor the author may enter the code to be executed when the timed event is triggered see the sample below.

Sample Usage:

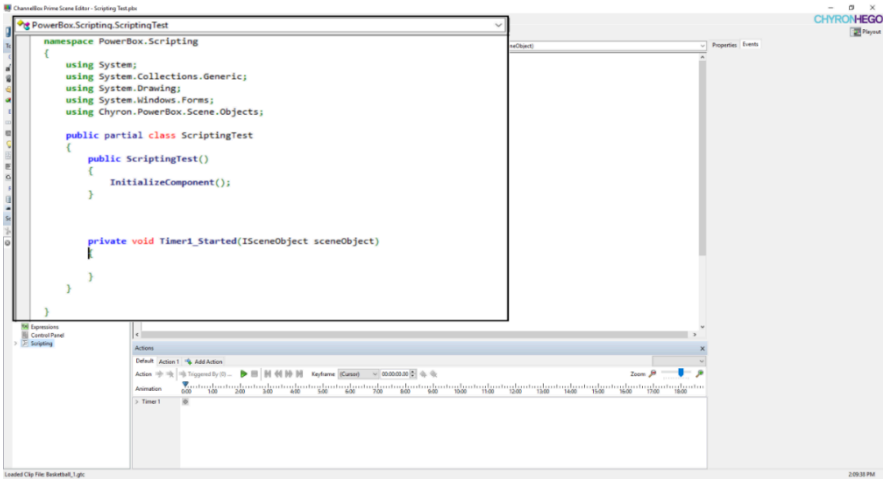
```
private void Timer1_Finished(ISceneObject sceneObject)
{
    // Your code here
}
```

Started Event

The scene author can add a Started event through the events tab that gets triggered when the timer starts.



Clicking on the Started drop down box a list of items that can be triggered is presented. Check the Timer1_Started item under Scripting and the **Timer1_Started()** method will be automatically added to your scene script.



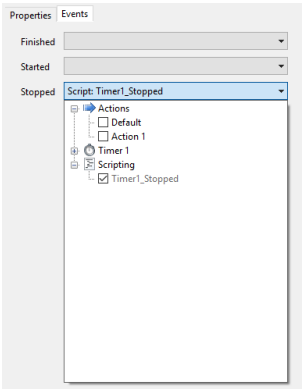
From the script editor the author may enter the code to be executed when the timed event is triggered see the sample below.

Sample Usage:

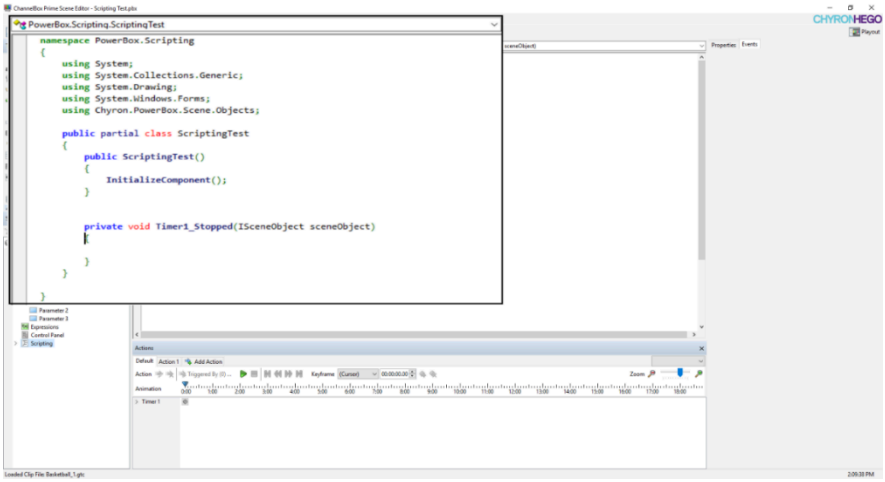
```
private void Timer1_Started(ISceneObject sceneObject)
{
    // Your code here
}
```

Stopped Event

The scene author can add a Stopped event through the events tab that is triggered when the timer is stopped.



Clicking on the Stopped drop down box a list of items that can be triggered is presented. Check the Timer1_Stopped item under Scripting and the **Timer1_Stopped()** method will be automatically added to your scene script.



From the script editor the author may enter the code to be executed when the timed event is triggered see the sample below.

Sample Usage:

```
private void Timer1_Stopped(ISceneObject sceneObject)
{
    // Your code here
}
```

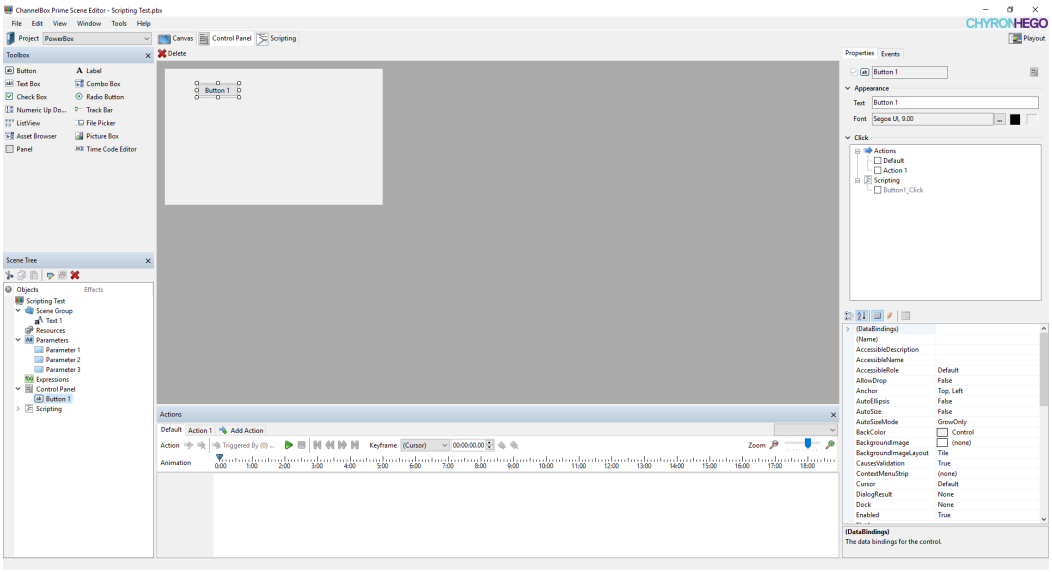
Button Events

The button allows the ***click*** event to be handled by the scene author through scripting. Below are the various methods for adding a button to the control panel so that the click event template code is automatically generated and added to the C# scripting code.

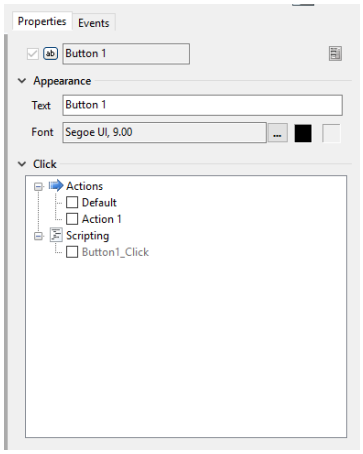
Toolbox

To add a new button to the control panel the scene author can click the 

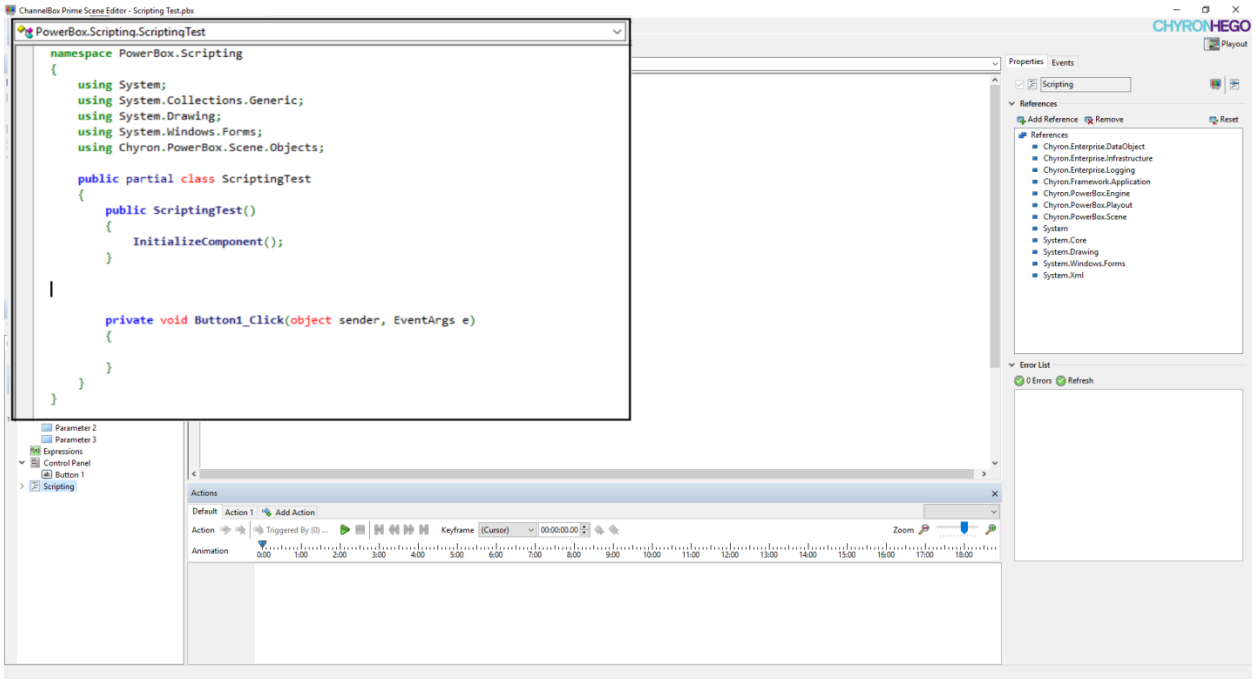
button in the toolbox list that appears on the left hand side of the control panel. Once added you will see the Button properties on the right hand side of the control panel as seen below



The button properties available to the scene author are shown below. We are interested in the **Click** section, which contains the list of item that can be triggered when the button is pressed.



Check the Button1_Click item under the Scripting section and the system will automatically add the **Button1_Click()** method to you scenes script.



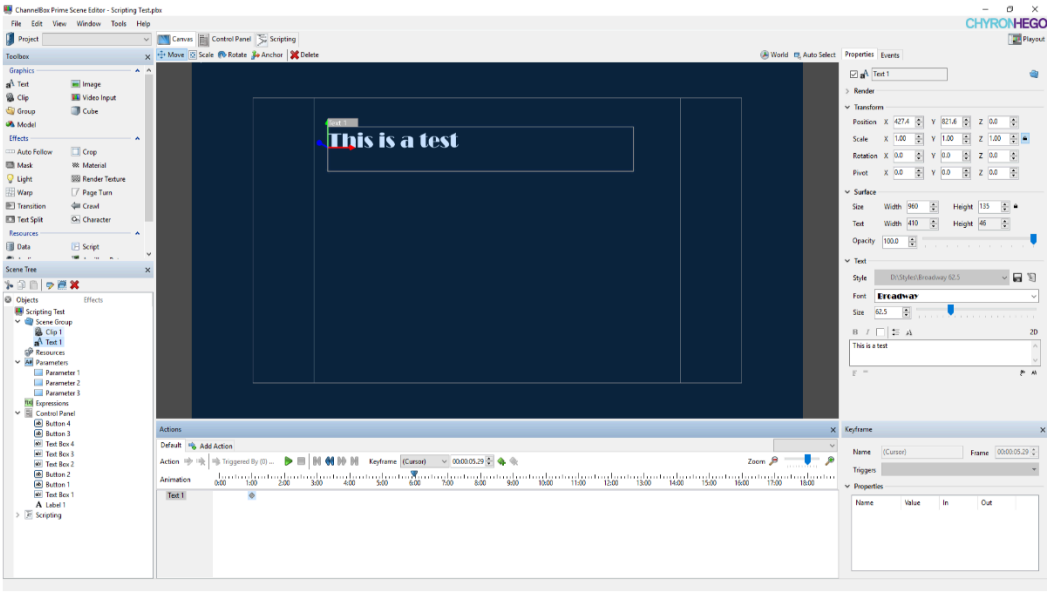
From the script editor the author may enter the code to be executed when the button is clicked see the sample below.

Sample Usage:

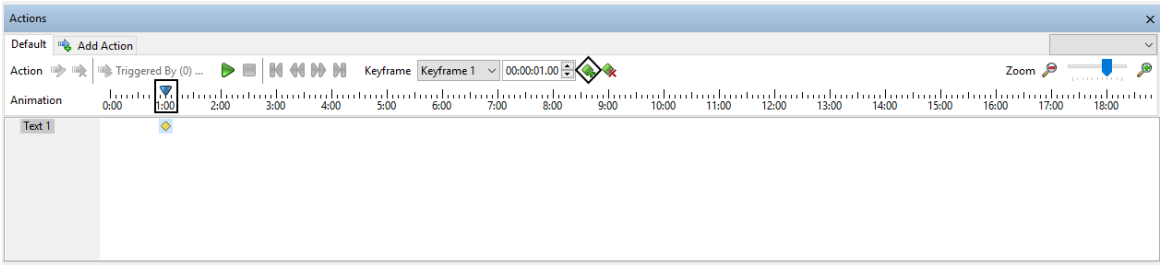
```
private void Button1_Click(object sender, EventArgs e)
{
    // Add your code here

}
```

Keyframe Trigger

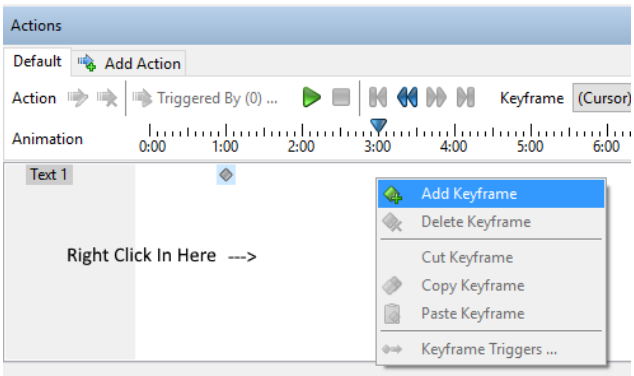
The keyframe triggers are available from the designer editor for all graphic objects.



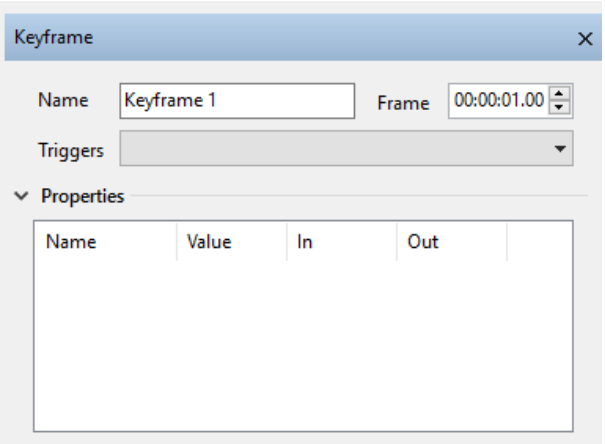
To add a new **keyframe** that can be bound to a trigger you first move the timeline tracking icon  to the time offset you want the **keyframe** added then click the add icon  . These icons are highlighted below.



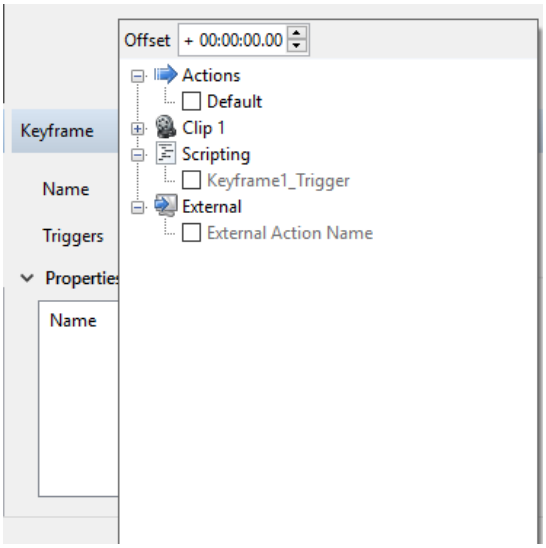
You also can right click in the area below the timeline to bring up the context menu and select *Add Keyframe* option.



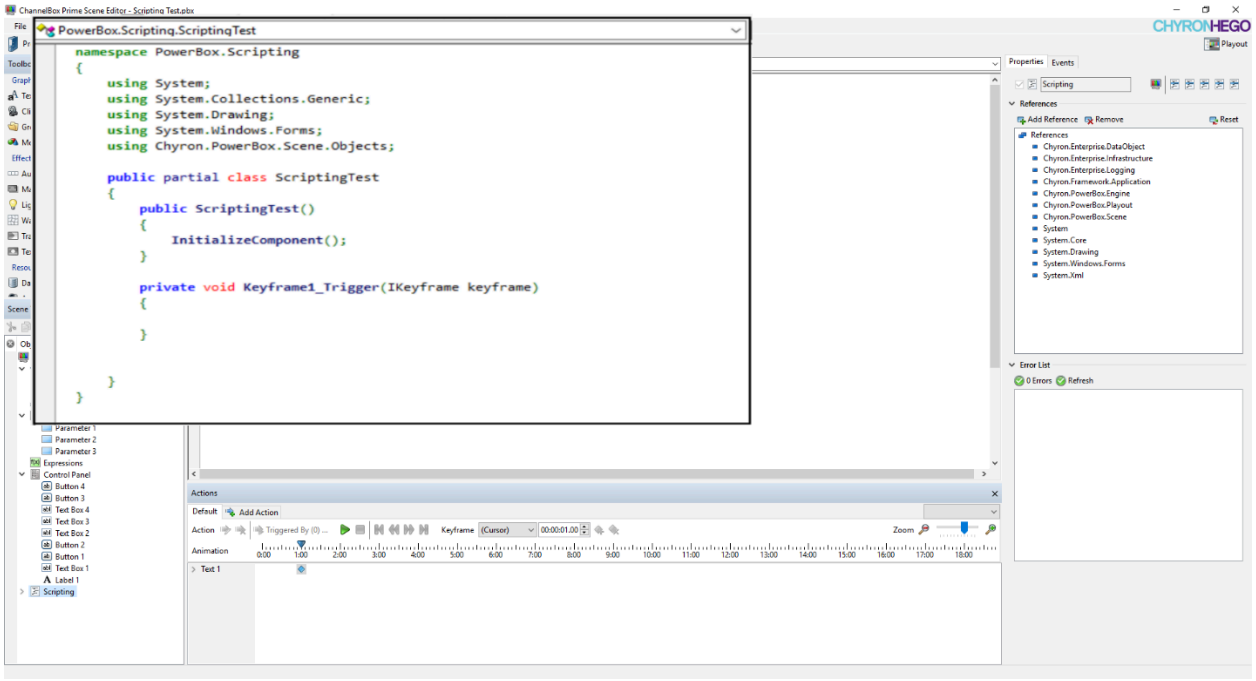
Once the **keyframe** has been added you will see over in the properties section below the ability to bind this **keyframe** to a trigger by clicking the **Triggers** drop down button.



The drop down will display all the events that the keyframe can trigger. We are interested in the Scripting option were **KeyFrame1_Trigger** can be checked or unchecked.



Checking off the **KeyFrame1_Trigger** selection will automatically add the **KeyFrame1_Trigger()** event handler method into the scripting code.



From the script editor the author may enter the code to be executed when the keyframe is triggered see the sample below.

Sample Usage:

```
private void Keyframe1_Trigger(IKeyframe keyframe)
{
    // Add your code here
}
```

Interface

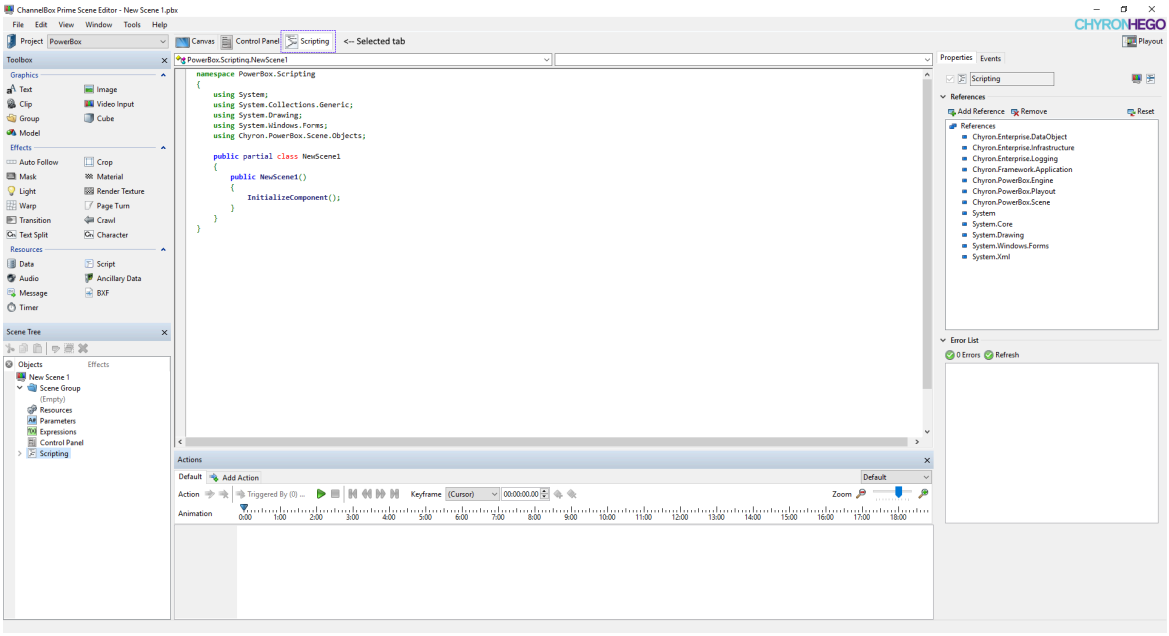
There are two types of C# scripting available in **PRIME**. Application scripting is a single C# script that has an application wide scope. Scene scripting has a scope of the scene. Each scene can have its own C# script, but there is only one application script. Both scripts share many design attributes with only a few minor differences, which will be discussed.

Scripting Editor

Scene Scripting

The C# scripting editor for the scene is available through the designer by clicking the 

toolbar button, which will display the C# scripting editor shown below.



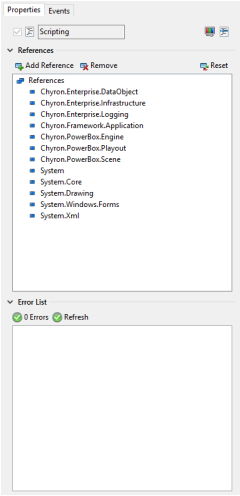
The Editor pane is what the scene author will be working in while writing the C# script. Upon startup for the first time the pane will contain the default code template (see below) for creating the C# script.

Default Scene Scripting Code Template (Scene Name: NewScene)

```
namespace PowerBox.Scripting
{
	using System;
	using System.Collections.Generic;
	using System.Drawing;
	using System.Windows.Forms;
	using Chyron.PowerBox.Scene.Objects;

	public partial class NewScene1
	{
		public NewScene1()
		{
			InitializeComponent();
		}
	}
}
```

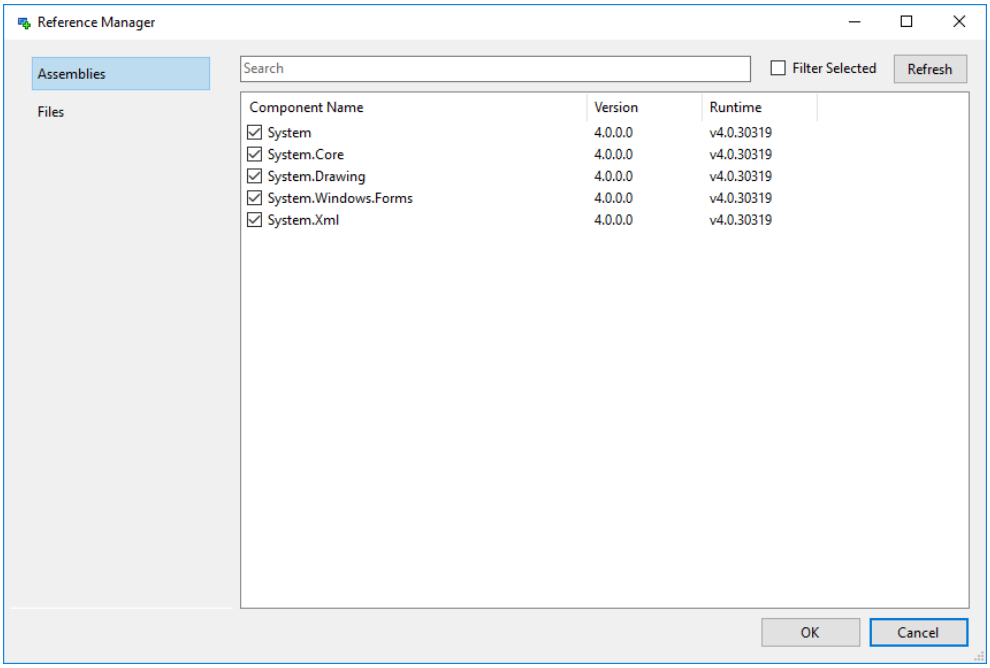
Scripting Properties

To the right of the scripting panel is the properties for the script, which shows the assembly references and error list for the code indicting syntax errors.



References

The **References** pane lists all the assembly references used in the script. There will be the standard Microsoft.NET assemblies as well as PRIME assemblies. The scene author can remove references from the list by selecting them and clicking the Remove button from the references toolbar. To add new references, the scene author can click the Add Reference button on the references toolbar, which will launch the **Reference Manager**.

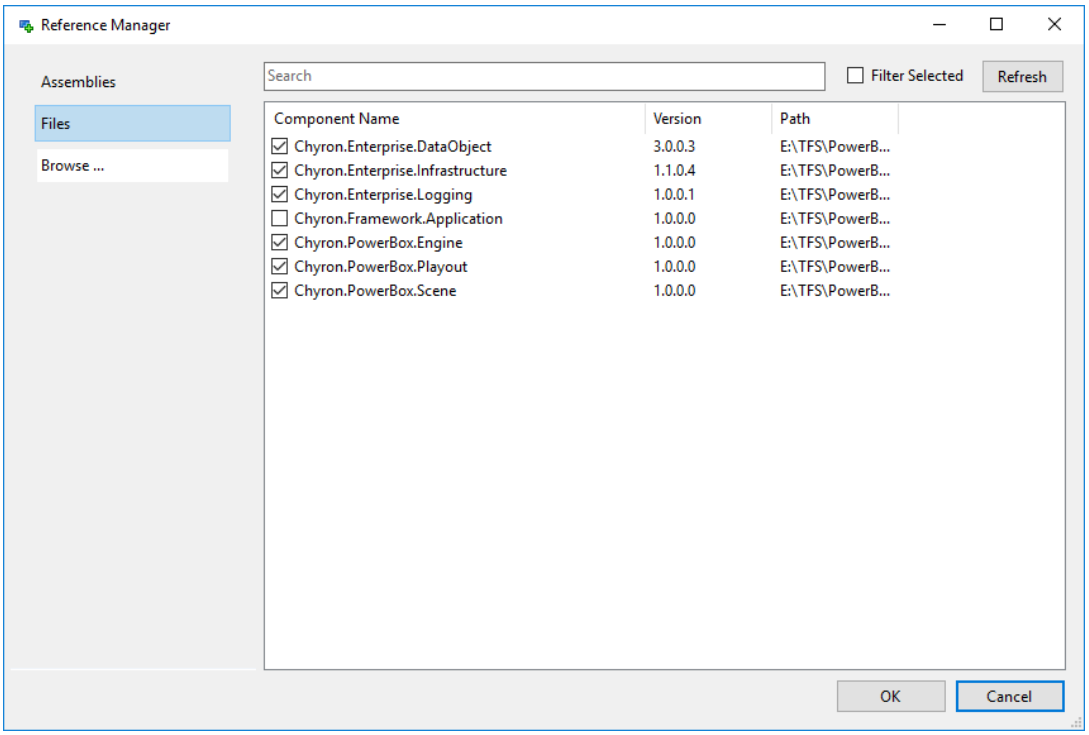


Assemblies

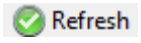
The **Assemblies** option, which is the default selection and will display all system wide Microsoft.NET assemblies installed on the system. Any assembly that has already been added to you script will appear with a check and any assembly that can be added will appear unchecked. Checking any unchecked entry will add the reference to your C# script and allow access to any classes available in the selected assembly. Unchecking any assembly will result in that assembly being removed from your script and removing access to any classes available from that assembly. Any assemblies added and/or removed through **Reference Manager** will be reflected in the References pane of the C# scripting editor and will be visible when the manager is closed.

Files

The **File** option if selected will display all the PRIME assemblies, as well as any additional assemblies that the user may have loaded. The first time the user will only see the pre-selected **PRIME** assemblies based on the default code template. To add new assemblies that do not already appear on the list you can click the **Browse …** button to launch the standard Windows file browse dialog. Any newly added assemblies from the file browser will be automatically checked and added to the scripts reference list. Unchecking any assembly will result in that assembly being removed from your script and removing access to any classes available from that assembly. Any assemblies that are displayed in red text will indicate that the assembly dll cannot be found.

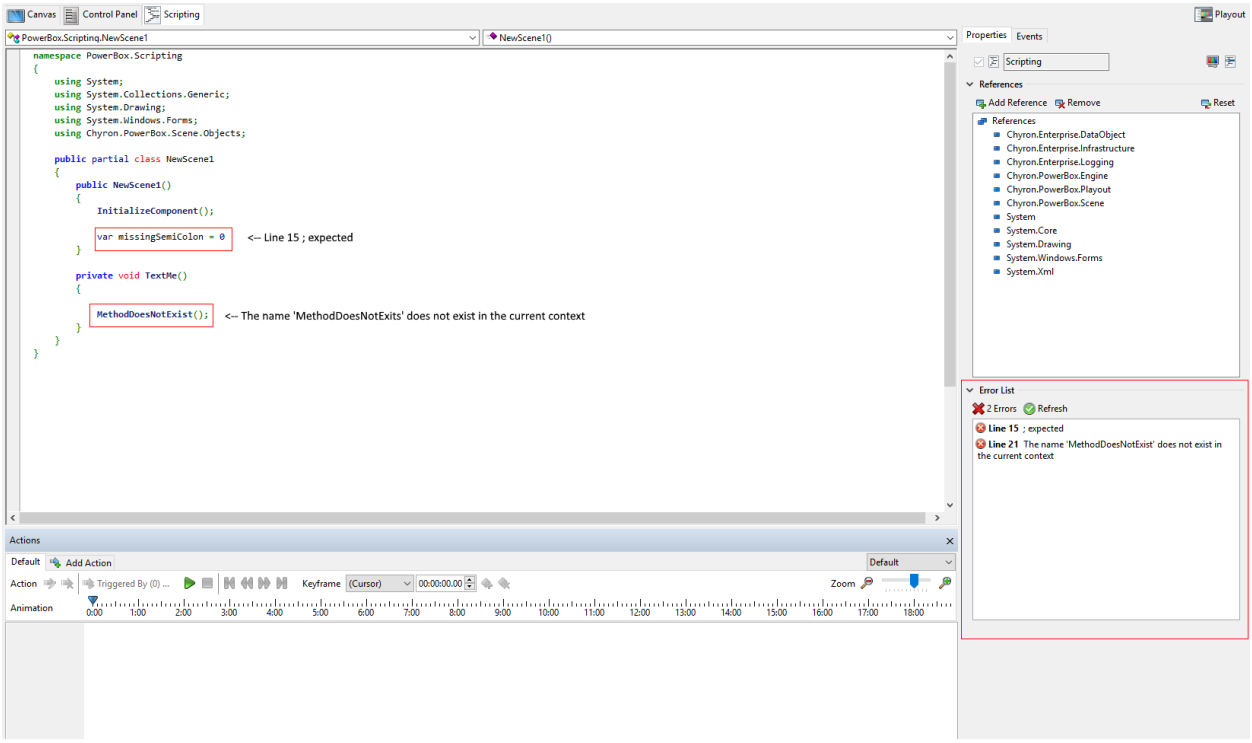


Error List

The Error List pane will display any errors that are detected in the script file. Some errors may occur when the script editor is loaded such as assembly loading errors. These errors indicate that the assembly referenced in the editor could not be loaded for some reason. By clicking on the 

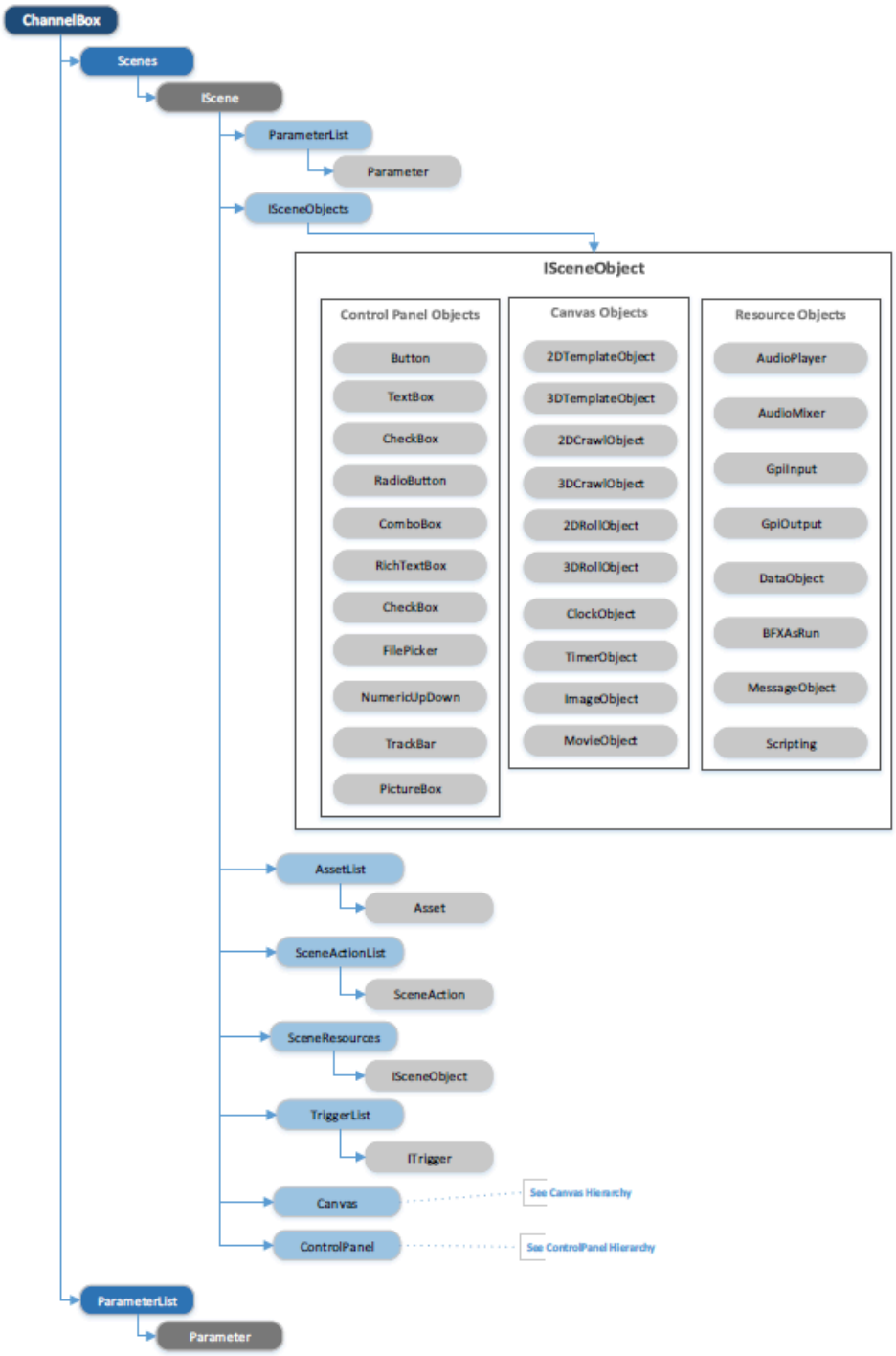
button on the toolbar the system will attempt to load the references, check the syntax of the C# script display any errors in the **Error List** and indicate the number of errors 

.



From the errors shown above, it indicates that a “; is expected” on line 15 in the C# script and that **MethodDoesNotExist()** does not exist in the context. The scene author can double click on the error and it will put the cursor at the beginning of the script line that caused the error.

Scripting Properties



Appendix

Text File Reader

Scene allows the user to select a file to be read in line by line and displayed on the Canvas using a graphic Text object. Action is added to have a fade in and fade out effect.

Canvas

Object	Name	Description
Text	MyTextObject	Used to display text from the file.

Control Panel

Object	Name	Description
Button	StartButton	Used to start and stop the file reading. **Events** **Click** **StartButton_Click**
FilePicker	TextFilePicker	Used to select file from the file system. **Events** **FileChanged** **TextFilePicker_FileChanged**
TextBox	LineTextBox	Used to display the text that is being displayed in Canvas. **Binding** **LineTextObject.Text**
Action	FadeAction	Used to have the text fade in and fade out. **Trigger** **LineTextObject.TextChanged**

Script

To get the file to be read each time the action is done using a Timer object by setting the interval to the FadeAction's length in milliseconds. The timer is then set to AutoReset so that after an Elapsed event is triggered it will restart automatically. We register the OnTimerElapsed() event handler to the timers Elapsed event but we don't start the timer yet. The file to be read is selected by using the FilePicker control on the control panel. Once a valid file is selected, clicking button will open a file stream for reading, read the first line, start the timer and change the buttons text to "Stop". Clicking the button when is read "Stop" will stop the timer, close the file and change the buttons text back to "Start".

```
namespace PowerBox.Scripting
{
   using System;
   using System.Collections.Generic;
   using System.Drawing;
   using System.Windows.Forms;
   using Chyron.PowerBox.Scene.Objects;
   using Chyron.Enterprise.Infrastructure.Parameters;
   using System.IO;
   using System.Timers;

   public partial class TextFileReader
   {

        private string filePath;
        private System.Timers.Timer timer;
        private StreamReader reader;

        public TextFileReader()
        {
          InitializeComponent();

          StartButton.Enabled = false;

          timer = new System.Timers.Timer((double)(Action1.Length.Seconds*1000));
          timer.AutoReset = true;

          timer.Elapsed += OnTimerElapsed;
        }

        // Handles the Elapsed event from the timer
        private void OnTimerElapsed(object sender, ElapsedEventArgs e)
        {
          ReadNextLine();
        }

        //Handles the FileChanged event from the FilePicker
        private void TextFilePicker_FileChanged()
        {
          TextFilePicker.ForeColor = Color.Black;

          filePath = TextFilePicker.File;

          StartButton.Enabled = ( !string.IsNullOrWhiteSpace(filePath) &&
                                  File.Exists(filePath) );
```

```
}


// Reads the next line from the file and updates the LineTextObject's Text property
private void ReadNextLine()
{
  if ( reader != null )
  {
    if ( ! reader.EndOfStream )
    {
      var line = reader.ReadLine();

      LineTextObject.Text = line;
    }
  }
}

// Handles the Click event for the StartButton
private void StartButton_Click(object sender, EventArgs e)
{

  if ( StartButton.Text == "Start" )
  {

    if ( File.Exists(filePath) )
    {

       try
       {
              reader = new StreamReader(new FileStream(filePath, FileMode.Open));
       }
       catch ( Exception ex )
       {
              TextFilePicker.ForeColor = Color.Red;
              this.Scene.LogError(ex.Message);
              return;
       }

    }

    ReadNextLine();

    timer.Start();

    StartButton.Text = "Stop";
  }
  else
  {
    timer.Stop();

    if ( reader != null )
    {
       reader.Close();
       reader = null;
    }

    StartButton.Text = "Start";
```

```
        }
    }
  }
}
```

Side Show

Scene allows the user to select a file that contains a list of images or a directory that contains images to be read one at a time and display the image on the canvas. A fade in / fade out action on the Canvas using a graphic Text object. Action is added to have a fade in and fade out effect.

Canvas

Object	Name	Description
Image	SlideImage	Used to display the image on the canvas.
Text	SlideImageName	Used to display the image name on the canvas.

Control Panel

Object	Name	Description
Button	ButtonSlideShow	Used to start and stop the slide show. **Events** **Click** **ButtonSlideShow_Click**
TextBox	ImageFileTextBox	Use to display the image filename.
TextBox	ImageFolderTextBox	Used to enter or display the folder to search for images. **Events** **TextChanged** **ImageFolderTextBox_TextChanged**
FilePicker	ImageListFilePicker	Used to select the file containing a list of images to be used,

		Events **FileChanged** **ImageListFilePicker_FileChanged**
Button	BrowserButton	Used to launch the folder browser to select the directory to search for images. **Events** **Click** **BrowserButton_Click**
Action	FadeAction	Used to have the image fade in and fade out. **Trigger** **SlideImage.FileChanged**
Parameter	ButtonStartText	Contains the text to be displayed on the ButtonSlideShow to indicate it will Star the slide show.
Parameter	ButtonStopText	Contains the text to be displayed on the ButtonSlideShow to indicate that it will Stop the slide show.
Parameter	ImageExtensions	Contains a commas delimited list of valid image file extensions.

Script

This script uses some advance features such as threading to perform the actions required. The image list is created as a stack so that it is processed only once as each item is popped off and displayed. The list is created when the StartButton is clicked and the Tag property is equal to SlideShowOptions.Start enumeration value. The control panel's controls that are used by the slide show process are disabled so that entries cannot be changed while the list is being processed. The image list is either generated using the image text file, which contains a list of images and their paths or a directory that contains images. A separate thread is started to perform the read so that other actions can be done without interruption. When the list of images has been exhausted the control panels state is set back so that a new slide show can be performed. At any point, while the image list is being processed the StartButton may be clicked to Stop and reset the control panel back.

```
namespace PowerBox.Scripting
{
    using System;
    using System.Collections.Generic;
    using System.Drawing;
    using System.Windows.Forms;
    using Chyron.PowerBox.Scene.Objects;
    using Chyron.PowerBox.Scene.Controls;
    using Chyron.PowerBox.Scene.Text;
    using System.Linq;
    using System.IO;

    public partial class SlideShow
    {

        private enum SlideShowOptions
        {
               Start,
               Stop
        };


        private bool allfiles = false;
        private string filePath;

        private bool cancel = false;
        private Stack<string> imageList = new Stack<string>();
        private string startText = "Start Slide Show";
        private string stopText = "Stop Slide Show";
        private string [] extensions = new string [0];
        private System.Windows.Forms.FolderBrowserDialog folderBrowser =
                              new System.Windows.Forms.FolderBrowserDialog();

        public SlideShow()
        {
          InitializeComponent();

          ButtonSlideShow.Enabled = false;

          InitializeScript();


        }

        // initialize the script
        private  void InitializeScript()
        {

          var parameter = Scene.Parameters.Find("ButtonStartText");
          if ( parameter != null )
          {
               startText = parameter.Value as string;
          }

          parameter = Scene.Parameters.Find("ButtonStopText");
          if ( parameter != null )
          {
               stopText = parameter.Value as string;
```

```
  }

  extensions = ImageExtensions.Value.ToLower().Split(new char[] { ',' });

  allfiles =  extensions.Contains("*") || extensions.Contains("*.*");
  ButtonSlideShow.Text = startText;
  ButtonSlideShow.Tag = SlideShowOptions.Start;

  ImageFileTextBox.Text = "";
  Scripting.File = "";
  Scripting.Opacity = 0;

}

// Creates the image list from either the directory or the image list file
private void CreateImageList()
{
  imageList.Clear();

  BrowseButton.Enabled = f
  ImageFileTextBox.Text = "Creating Image List";

  if ( File.Exists(filePath) )
  {
    using ( StreamReader  reader = new StreamReader(filePath) )
    {
       while ( !reader.EndOfStream )
       {
         var line = reader.ReadLine();

         AddImageFile(line);
       }
    }

  }
  else if ( Directory.Exists(filePath) )
  {
    foreach ( var file in Directory.GetFiles(filePath, "*.*") )
    {
       AddImageFile(file);
    }
  }
}

// Add image file to the stack
private void AddImageFile(string file)
{
  // image files extension
  var extension = System.IO.Path.GetExtension(file);

  // image file must exist and its extension must be allowed
  if ( File.Exists(file) && ( allfiles ||
             ( !allfiles && extension.Contains(extension) ) ) )
  {
    imageList.Push(file);
  }

}
```

```
// Enables the slide show
private void EnableSlideShow()
{

  // make sure we are executing on the GUI thread
  if ( this.InvokeRequired )
  {
    Invoke(new Action(EnableSlideShow));
  }
  else
  {

     SlideImageName.Text = "";
     Scripting.Opacity = 0;

     ImageFileTextBox.Text = "";

     ImageFileTextBox.Enabled = true;
     BrowserButton.Enabled = true;
     ImageListFilePicker.Enabled = true;

     ButtonSlideShow.Enabled = true;
     ButtonSlideShow.Text = startText;
     ButtonSlideShow.Tag = SlideShowOptions.Start;
     ButtonSlideShow.BackColor = Color.LightSteelBlue;

     Log("");
  }

}

// Handle the button click event for the slide show
private void ButtonSlideShow_Click(object sender, EventArgs e)
{

    var option = (SlideShowOptions)ButtonSlideShow.Tag;

    if ( option == SlideShowOptions.Start )
    {


       CreateImageList();

       if ( imageList.Count > 0 )
       {
          cancel = false;

          ButtonSlideShow.BackColor = Color.Red;

          ImageFileTextBox.Enabled = false;
          BrowserButton.Enabled = false;
          ImageListFilePicker.Enabled = false;

          ButtonSlideShow.Text = stopText;
```

```
            ButtonSlideShow.Tag = SlideShowOptions.Stop;


            System.Threading.Tasks.Task.Factory.StartNew(()=>
            {
                while ( imageList.Count > 0 && !cancel )
                {
                  var image = imageList.Pop();
                  if ( System.IO.File.Exists(image) )
                  {
                    ImageFileTextBox.Text = image;
                    Scripting.File = image;
                    SlideImageName.Text =
                         System.IO.Path.GetFileNameWithoutExtension(image);
                   }


                   System.Threading.Thread.Sleep(FadeAction.Length.TimeSpan);


                }

                if ( cancel )
                {
                  FadeAction.Stop();

                  Log("Slideshow cancelled");
                }
                else
                  Log("Slideshow done");


                EnableSlideShow();

            });
          }

      }
      else
      {
        ButtonSlideShow.BackColor = Color.Gray;
        ButtonSlideShow.Enabled = false;
        cancel = true;

      }

    }

    // Handles the TextChanged event for the ImageFolderTextBox control
    private void ImageFolderTextBox_TextChanged(object sender, EventArgs e)
    {
      if ( Directory.Exists(ImageFolderTextBox.Text) )
      {
        filePath = ImageFolderTextBox.Text;
        ImageListFilePicker.File = "";
        ButtonSlideShow.Enabled = true;
      }
      else
```

```
        {
          if ( File.Exists(ImageListFilePicker.File) )
          {
            ButtonSlideShow.Enabled = true;
          }
          else
          {
             ButtonSlideShow.Enabled = false;

             filePath = "";
           }
        }

      }

      // Handles the Click event for the BrowseButton control
      private void BrowserButton_Click(object sender, EventArgs e)
      {
        folderBrowser.Description = "Select Image Directory";
        folderBrowser.ShowNewFolderButton = false;

        if ( folderBrowser.ShowDialog(this) == System.Windows.Forms.DialogResult.OK)
        {
          ImageFolderTextBox.Text = folderBrowser.SelectedPath;
        }
      }

      // Handles the FileChanged event from the ImageListFilePicker control
      private void ImageListFilePicker_FileChanged()
      {
        if ( File.Exists(ImageListFilePicker.File) )
        {
          filePath = ImageListFilePicker.File;
          ImageFolderTextBox.Text = "";
          ButtonSlideShow.Enabled = true;

        }
        else
        {
          if ( Directory.Exists(ImageFolderTextBox.Text) )
          {
            ButtonSlideShow.Enabled = true;
          }
          else
          {
            ButtonSlideShow.Enabled = false;
            filePath = "";
          }

        }
      }
  }

}
```

VB-JSCRIPT

PRIME also includes support for VBScript and JScript. The common API can be utilized to interact with and control the playback of scenes.

The Model

There is only one PRIME application instance running on a system at a time. The application services each request through the API, providing the ability to retrieve and modify the state of scenes within the currently selected project.

The **Scene** is the basic PRIME structure, containing multiple graphical and hardware elements called **Objects** as well as **Controls** that define an interface to configurable aspects of the scene. Scenes also contain one or more **Actions**, which define object animations, and may be triggered through the scripting API. Currently the actions do not expose their underlying keyframes and associated properties, though this may be added in future revisions of the API.

Each scene also has a collection of customizable **User Parameters**, which allows scripts to store and retrieve context information at runtime.

Script Execution and Context

Scripts associated with a scene at design time are executed in-process. In-process scripts are provided context and a handle on the PRIME COM API in the form of global keywords that are accessible from the script (see Global Keywords).

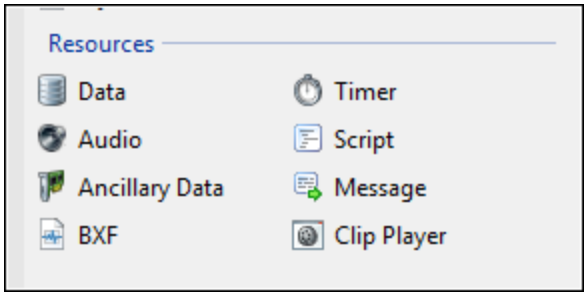
When PRIME is running with administrator privileges, scripts may also be executed out of process (e.g. by double-clicking a VBS file on the Windows desktop or from a VBScript editor), although these scripts must tap into the PRIME COM API differently since they have no local context:

```
Dim Prime
Set Prime = CreateObject("Prime", "localhost")
```

Future revisions of scripting may include a DCOM implementation so that scripts will have access to the PRIME API from any COM environment even if the PRIME application has not been started.

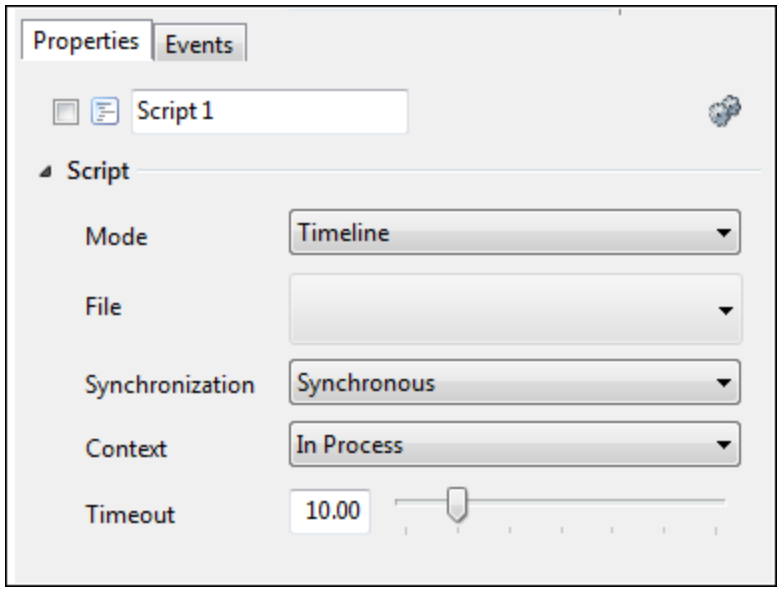
Scene Designer

The user may incorporate scripts using VBScript or JScript by adding a Script resource to their scene. The resource can be added by clicking the Script button in the Designer Toolbox as seen below:



Once added, the Script resource may be customized in a variety of ways, but the most important configuration decision is deciding how script execution will occur. The **Mode** property dictates whether scripts will execute when keyframes animate or in response to designated scene events.

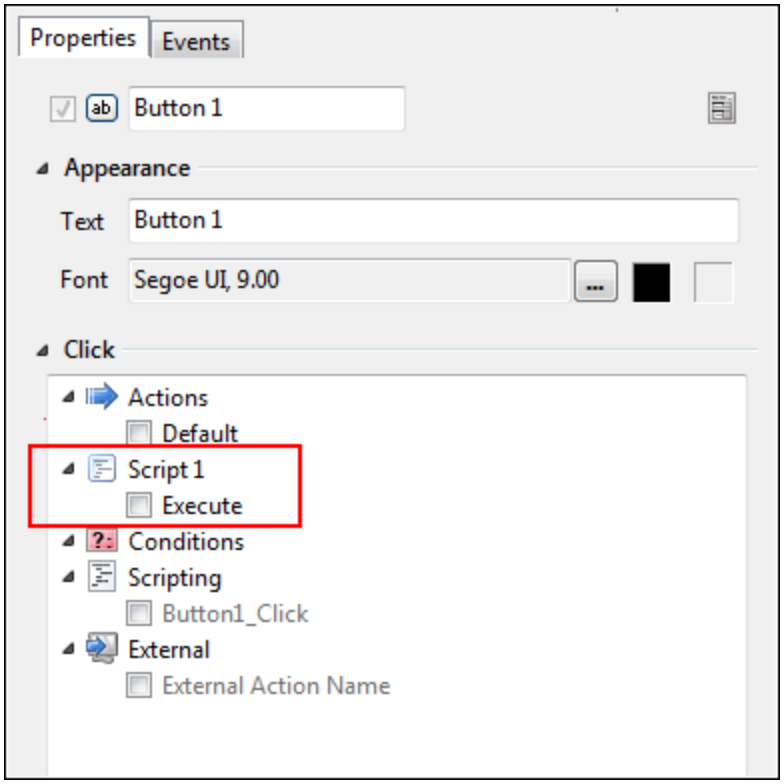
Script resources default to *Timeline* mode, indicating that the object will execute the selected file at designated keyframes on the timeline. Each of the other properties listed are keyframeable; consequently, a single Script resource may be used to execute multiple script files: each keyframe will a different file property affected.



- **File**: The VBScript or JScript file that will be executed. Browsing for a file will initially suggest files local to the current project, but the user may opt to select a completely external file.
- **Synchronization**: This property specifies whether the script will be executed synchronously or asynchronously. Synchronous script execution can impede the execution of other scripts however this may be preferable and is the default behavior. Asynchronous script executions may occur concurrently.

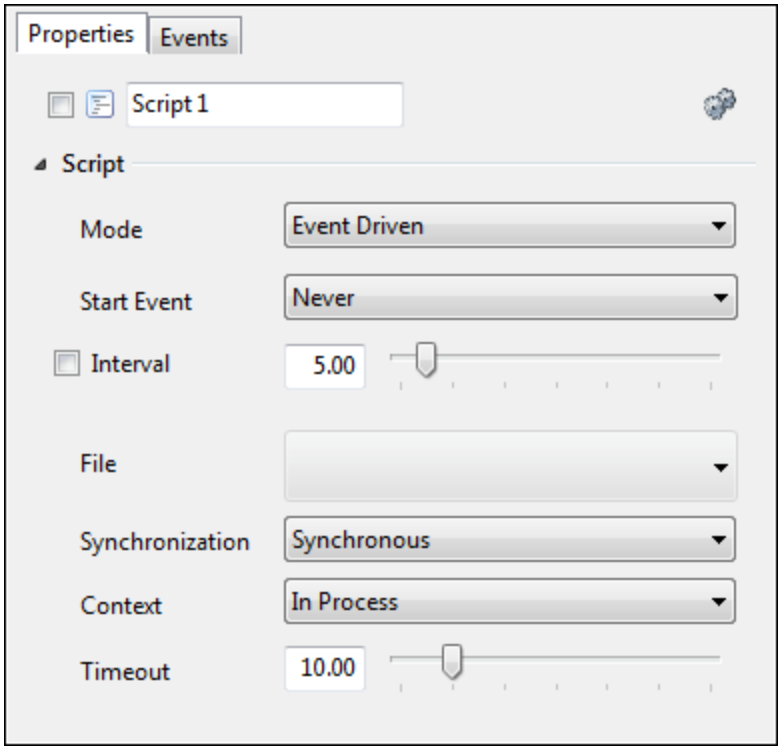
- **Context**: Indicates whether the script will be executed *In Process* (with context of the executing scene) or *Out of Process* (no context). Generally, scripts that will interact with the PRIME API should be executed *In Process*. Scripts that don't require the PRIME API, such as those that download data on some interval should be executed *Out of Process*.
- **Timeout**: Indicates how long a script should allow for execution before timing out (and forcibly terminated). This property is used to prevent long-running scripts from interfering with playout operation or performance.

Scripts in this mode may also be triggered in the standard PRIME trigger list control as typically seen when configuring object or scene events. For example, a button control can be added and configured to execute a Script resource as seen below.

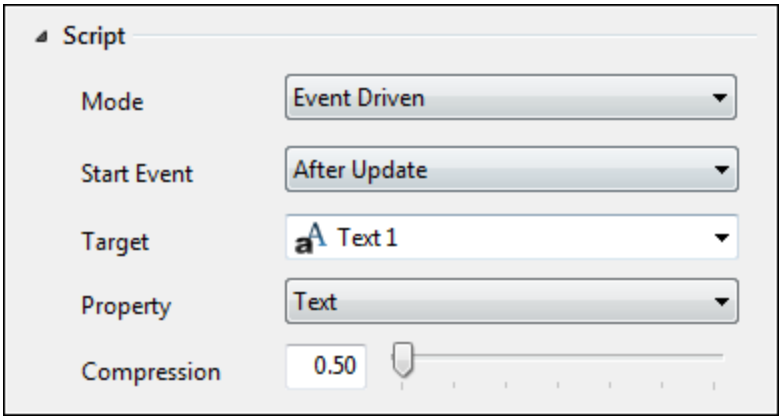


This would execute the script file as configured on the default keyframe of the resource.

Alternatively, the **Mode** property can be changed to *Event Driven*. Additional properties become available when this value is set.



- **Start Event**: Indicates the type of event that will cause the script to execute.
    - *Never*: Script execution is disabled.
    - *After Close*: The script will execute after the scene is closed.
    - *After Load*: The script will execute after the scene is loaded.
    - *After Play*: The script will execute after the scene is taken to air.
    - *After Stop*: The script will execute after the scene is transferred off air.
    - *After Update*: The script will execute after a specific property of a particular object has changed.

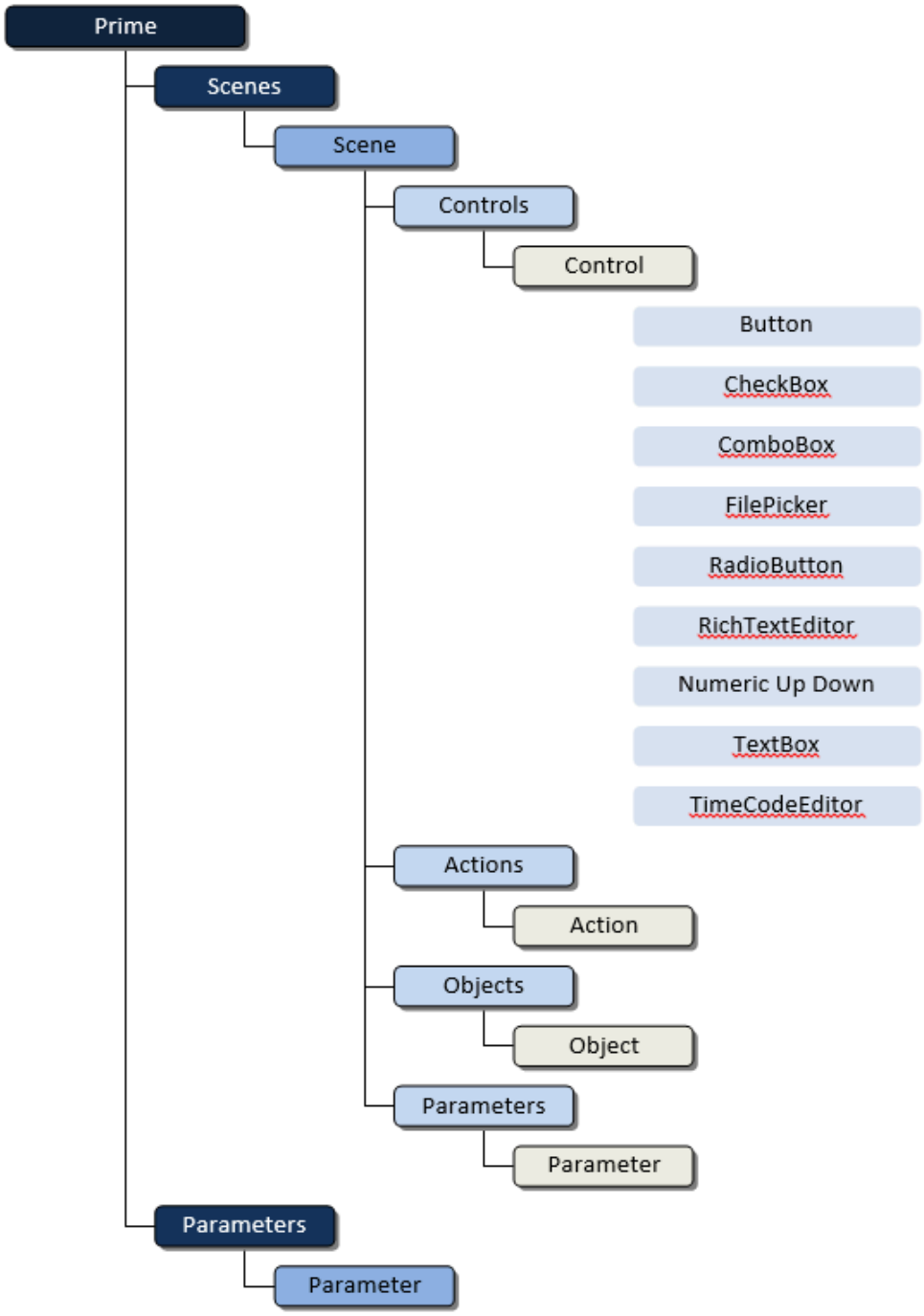


    - *Before Close*: The script will execute before the scene is closed.

    - *Before Play*: The script will execute before the scene is taken to air.
    - *Before Stop*: The script will execute before the scene is transferred off air.
    - *Before Update*: The script will execute before a specific property of a particular object has changed. The executing script may change the value prior to its application.
    - *While Playing*: The script will execute only when the scene is on air.

- **Interval**: If enabled, specifies that the script should execute repeatedly on a defined interval (in seconds). For example, the user could configure a script that executes every five seconds while on air by setting the **Start Event** to *While Playing* and the checking the Interval property.

Object Hierarchy



Global Keywords

Below are reserved keywords that may be used by an in-process script to reference the PRIME API.

ActiveScene

Returns the scene currently executing the script.

Sample Usage:

```
ActiveScene.Play ' Play the current scene
```

Prime

Returns a top-level object that can be used to interact with the currently running PRIME. This object is particularly useful for cross-scene scripting, meaning that a script in one scene can affect the state of other scenes.

Sample Usage:

```
Dim scene
Set scene = Prime.Scene("1") ' Active scene might not be 1

scene.Play
```

Parameters

Returns a collection of script parameters that contains useful information about the script being executed. Currently, only four parameters are possible: **Control Name**, **Control Value**, **Id** and **Script Name**.

Whenever a script is executed from PRIME, the script is provided with a collection of parameters that may be of use to the executing script. Under most situations, this is currently limited to the identifier of the scene (*Id*) that caused execution of the script as well as the name associated with the executing script (*Script Name*), however, when a script executes as a result of an update to a control panel control, two additional parameters are made available.

Script Parameter	Example VBScript Usage	Available When?	Read only?
Id	Parameter.Item("Id")	Always	Yes
Control Name	Parameter.Item("Control Name")	Before, After Update	Yes

Control Value	Parameter.Item("Control Value")	Before, After Update	No
Script Name	Parameter.Item("Script Name")	Always	Yes

It may be desirable to execute a script that can transform an incoming data value before applying the value to a control panel control. This can be accomplished by adding a Script resource to the scene in *Event Driven* mode, marking the *Execute Script* property *Before Update* and setting the Target property to the desired control. At this point, the script will execute immediately before a value is applied to the target control, allowing the script author to intercept the value and make any desired modifications.

Modifying the incoming value requires that the *Control Value* parameter is used. Simply set the value of the **Script Parameter** as seen below:

```
Parameter.Item("Control Value") = "New Value"
```

In the example above, *New Value* will be applied to the control after the script completes executing instead of the original value.

API Reference

Prime

The global keyword **Prime** returns an application proxy, which provides hooks into the PRIME API. This is the top level object that allows a script author to interact with scenes in the currently selected project.

AllScenes

Returns a collection of all scenes in the currently selected project.

Sample Usage:

```
Dim scene
' Iterate through each scene
For i = 0 to Prime.AllScenes.Count - 1
    ' Here we can do something with each scene.
    Set scene = Prime.AllScenes.Item(i)
    ' For example, show a message box with the scene ID.
    MsgBox scene.SceneId
Next
```

OpenScenes

Returns a collection of all scenes that are currently open in the selected project.

Sample Usage:

```
Dim scene
' Iterate through each open scene
For i = 0 to Prime.OpenScenes.Count - 1
    ' Here we can do something with each scene.
    Set scene = Prime.OpenScenes.Item(i)
    ' For example, show a message box with the scene ID.
    MsgBox scene.SceneId
Next
```

Scene(id As String)

Returns a scene with the specified name, if one can be found. If the scene exists in the project, then that scene will be returned. If a matching scene cannot be found, then **Nothing** is returned.

Sample Usage:

```
Dim scene
Set scene = Prime.Scene("1")

' Check if the scene was returned
If Not(scene is Nothing) Then
    MsgBox "The scene was returned!"
Else
    MsgBox "Failed to get scene!"
End If
```

SceneExists(id As String)

Returns true if a scene with the specified name exists in the current PRIME project. Otherwise, returns false.

Sample Usage:

```
If Prime.SceneExists("1") Then
    MsgBox "The scene was found!"
Else
    MsgBox "The scene was not found!"
End If
```

SceneCollection

A collection of scenes returned by either **Prime.AllScenes** or **Prime.OpenScenes**.

Count

The number of scenes in the collection.

Sample Usage:

```
' Display the number of open scenes
MsgBox Prime.OpenScenes.Count
```

Item(index)

Returns a scene at the specified index in the collection.

Sample Usage:

```
Dim scene
' Iterate through each scene
For i = 0 to Prime.AllScenes.Count - 1
    ' Here we can do something with each scene.
    Set scene = Prime.AllScenes.Item(i)
    ' For example, show a message box with the scene ID.
    MsgBox scene.SceneId
Next
```

CloseAll

Closes all scenes in the scene collection.

Sample Usage:

```
Prime.AllScenes.CloseAll
```

StopAll

Stops all scenes in the scene collection.

Sample Usage:

```
Prime.AllScenes.StopAll
```

Scene

Although scripting in PRIME can be used without involving the PRIME API, perhaps the most powerful use of the feature is for interacting with scenes in the project. Use of the PRIME API allows a script author the ability to affect the state of scenes, store and retrieve context information for later use by other scenes or scripts, and a plethora of other features.

Scenes may be referenced by the script author in one of two ways.

1. The global **ActiveScene** keyword returns the scene that is currently executing the script. This keyword may be used anywhere in the script.

```
ActiveScene.Play ' Play the current scene
```

2. The global **Prime** keyword can be used to access any scene in the Prime database.

```
Prime.Scene("1").Play ' Play scene with ID 1
```

Close

Closes the scene, if the scene is not already closed.

Sample Usage:

```
Dim scene
Set scene = Prime.Scene("1")

scene.Close
```

Control(name)

Returns a control panel control with the specified name, if the scene's control panel contains one. If a matching control cannot be found, then **Nothing** is returned. An exception will be thrown if this property is accessed before the scene has been opened.

Sample Usage:

```
Dim control
Set control = ActiveScene.Control("Text Box 1")

' Check if the control was returned
If Not(control is Nothing) Then
    MsgBox "The control was returned!"
Else
    MsgBox "Failed to get control!"
End If
```

Controls

Returns a collection of all control panel controls found in the specified scene. An exception will be thrown if this property is accessed before the scene has been opened.

Sample Usage:

```
Dim scene, control
Set scene = ActiveScene

' Iterate through each control in the control panel
For i = 0 to scene.Controls.Count - 1
    ' Here we can do something with each control.
    Set control = scene.Controls.Item(i)
    ' For example, show a message box with the control name.
    MsgBox control.Name
Next
```

Load

Loads the scene regardless of its state.

Sample Usage:

```
Dim scene
Set scene = Prime.Scene("1")

scene.Load
```

Object(name As String)

Returns for an object with the specified name, if the scene contains one. If a matching object cannot be found, then **Nothing** is returned. An exception will be thrown if this property is accessed before the scene has been opened.

Sample Usage:

```
Dim sceneObject
Set sceneObject = ActiveScene.Object("Text 1")

' Check if the object was returned
If Not(sceneObject is Nothing) Then
    MsgBox "The object was returned!"
Else
    MsgBox "Failed to get object!"
End If
```

Objects

Returns an enumeration of all objects in the specified scene. An exception will be thrown if this property is accessed before the scene has been opened.

Sample Usage:

```
Dim scene, object
Set scene = ActiveScene

' Iterate through each object in the scene
For i = 0 to scene.Objects.Count - 1
    ' Here we can do something with each object.
    Set object = scene.Objects.Item(i)
    ' For example, show a message box with the object name.
    MsgBox object.Name
Next
```

Open

Opens the scene, re-opening the scene if it is already open.

Sample Usage:

```
Dim scene
Set scene = Prime.Scene("1")

scene.Open
```

Parameters

Returns a collection of user parameters currently associated with the scene. For additional details on working with parameters, see the Parameters section.

Play

Plays the scene regardless of its state. If the scene is already playing, the scene will first be stopped and then replayed.

Sample Usage:

```
Dim scene
Set scene = Prime.Scene("1")

scene.Play
```

SceneId

Returns the name of this scene. Aliasing functionality previously available in Channel Box is no longer supported.

Sample Usage:

```
Dim scene
Set scene = ActiveScene

MsgBox scene.SceneId
```

SceneState

Gets an integer indicating the current state of this scene.

- **2** if the scene is closed
- **4** if the scene is loaded
- **8** if the scene is loading currently
- **16** if the scene is opened
- **32** if the scene is playing

Sample Usage:

```
Dim scene
Set scene = Prime.Scene("1")

' Check if the scene is playing
If scene.SceneState = 32 Then
    ' Playing, so stop the scene
    scene.Stop
Else
    ' Not playing, so play the scene
    scene.Play
End If
```

Stop

Stops the scene, if it is playing.

Sample Usage:

```
Dim scene
Set scene = Prime.Scene("1")

scene.Stop
```

Control

Each scene has a set of controls that are collectively referred to as the control panel. These controls are exposed to scripting, providing specialized properties and functions pertinent to

each control type. All controls have two common properties, however, that can be used to identify the control and its supported functionality.

Name

Returns the name of this control.

Sample Usage:

```
Dim control
Set control = ActiveScene.Control("Button 1")

' Check if the control was returned
If Not(control is Nothing) Then
    MsgBox control.Name
Else
    MsgBox "Failed to get the control!"
End If
```

Type

Gets an integer indicating the type of this control. Currently labels, group boxes, and video proxies are unavailable through the API.

- **2** if the control is a button
- **3** if the control is a check box
- **4** if the control is a combo box
- **5** if the control is a file picker
- **10** if the control is a radio button
- **11** if the control is a rich text editor
- **12** if the control is a numeric up/down.
- **13** if the control is a text box
- **14** if the control is a time code editor

Sample Usage:

```
Dim scene, control
Set scene = ActiveScene

For i = 0 to scene.Controls.Count - 1
    Set control = scene.Controls.Item(i)

    ' Only display the names of TextBoxes
    If control.Type = 13 Then
        MsgBox control.Name
    End If
Next
```

Button

The Button can be used to interact with a control panel button.

Pressed

For state buttons, this property may be used to change or retrieve the current state of the button. True values correspond to the button being on, while false indicates that the button is off. For standard push buttons, this property may be used to simulate a button press by setting the value to true. This property will always return false for push buttons and setting its value to false should have no effect.

Sample Usage:

```
Dim button
Set button = ActiveScene.Control("Button 1")

If Not(button is Nothing) Then
    ' Check if state button is pressed
    If button.Pressed Then
        ' It is, so reset the button!
        button.Pressed = False
    End If
Else
    MsgBox "Failed to get the button!"
End If
```

CheckBox

The CheckBox can be used to interact with a control panel check box.

Checked

This property returns true if the check box is currently checked, otherwise false. The property may also be used to set the state of the check box.

Sample Usage:

```
Dim checkbox
Set checkbox = ActiveScene.Control("Check Box 1")

If Not(checkbox is Nothing) Then
    ' Check if state check box is checked
    If checkbox.Checked Then
        ' It is, so uncheck the checkbox!
        checkbox.Checked = False
    End If
Else
    MsgBox "Failed to get the checkbox!"
End If
```

ComboBox

The ComboBox can be used to interact with a control panel combo box.

ItemCount

Returns the number of items in the combo box drop down.

Sample Usage:

```
Dim combobox
Set combobox = ActiveScene.Control("Combo Box 1")

If Not(combobox is Nothing) Then
    MsgBox "There are " + combobox.ItemCount + " items in the combo box"
Else
    MsgBox "Failed to get the combo box!"
End If
```

Items

Returns an enumeration of the items in the combo box drop down. Each item corresponds to a text string visible in the drop down.

Sample Usage:

```
Dim combobox
Set combobox = ActiveScene.Control("Combo Box 1")
```

```
If Not(combobox is Nothing) Then
    For Each item in combobox.Items
        MsgBox "Item: " + item
    Next
End If
```

SelectedItem

Gets or sets the text string of the item currently selected in the combo box. You may only set a value using a text string that already exists in the combo box. All other updates to the selected item will be ignored.

Sample Usage:

```
Dim combobox
Set combobox = ActiveScene.Control("Combo Box 1")

If Not(combobox is Nothing) Then
    MsgBox "Selected Item: " + combobox.SelectedItem

    ' Now change the selected item
    ' This update only works if Item 2 exists in the combo box
    combobox.SelectedItem = "Item 2"
End If
```

GetItemValue(item)

Each item in a combo box may have an underlying value associated with it; for example, a combo box may display a shorter text string that may actually refer to a full file path. This function may be used to retrieve the actual value when given the visible item value. If this function is passed a text string that does not exist in the combo box, then **Nothing** is returned.

Sample Usage:

```
Dim combobox
Set combobox = ActiveScene.Control("Combo Box 1")

If Not(combobox is Nothing) Then
    MsgBox "Selected Item: " + combobox.SelectedItem

    ' Now display the actual value, which may be different
    MsgBox "Selected Item Value: " + combobox.GetItemValue(combobox.SelectedItem)
End If
```

FilePicker

The FilePicker can be used to interact with a control panel file picker.

Path

Gets or sets the file path currently in the file picker control.

Sample Usage:

```
Dim filepicker
Set filepicker = ActiveScene.Control("File Picker 1")

If Not(filepicker is Nothing) Then
    MsgBox "Current File Path: " + filepicker.Path

    filepicker.Path = "C:\Test.jpg"
Else
    MsgBox "Failed to get the filepicker!"
End If
```

RadioButton

The RadioButton can be used to interact with a control panel radio button.

Selected

This property returns true if the radio button is currently selected, otherwise false. The property may also be used to set the state of the radio button.

Sample Usage:

```
Dim radiobutton
Set radiobutton = ActiveScene.Control("Radio Button 1")

If Not(radiobutton is Nothing) Then
    ' Check if state radio button is selected
    If radiobutton.Selected Then
        ' It is, so reset the radiobutton!
        radiobutton.Selected = False
    End If
Else
    MsgBox "Failed to get the radio button!"
End If
```

RichTextEditor

The RichTextEditor can be used to interact with a control panel rich text editor.

Text

Gets or sets the text currently in the rich text editor control.

Sample Usage:

```
Dim texteditor
Set texteditor = ActiveScene.Control("Rich Text Editor 1")

If Not(texteditor is Nothing) Then
    MsgBox "Current Text: " + texteditor.Text

    texteditor.Text = "Hello, Goodbye"
Else
    MsgBox "Failed to get the text editor!"
End If
```

Numeric Up/Down

The Spinner can be used to interact with a control panel numeric up/down.

Value

Gets or sets the integer value currently in the numeric up/down control.

Sample Usage:

```
Dim spinner
Set spinner = ActiveScene.Control("Spinner 1")

If Not(spinner is Nothing) Then
    MsgBox "Current Value: " + spinner.Value

    spinner.Value = 100
Else
    MsgBox "Failed to get the spinner!"
End If
```

TextBox

The TextBox can be used to interact with a control panel text box.

Text

Gets or sets the text currently in the text box control.

Sample Usage:

```
Dim textbox
Set textbox = ActiveScene.Control("Text Box 1")

If Not(textbox is Nothing) Then
    MsgBox "Current Text: " + textbox.Text

    textbox.Text = "Hello, Goodbye"
Else
    MsgBox "Failed to get the textbox!"
End If
```

TimeCodeEditor

The TimeCodeEditor can be used to interact with a control panel time code editor.

Duration

Gets or sets the current value of the time code editor in the form “HH:MM:SS”.

Sample Usage:

```
Dim timeCodeEditor
Set timeCodeEditor = ActiveScene.Control("Time Code Editor 1")

If Not(timeCodeEditor is Nothing) Then
    MsgBox "Current Duration: " + timeCodeEditor.Duration

    timeCodeEditor.Duration = "10:20:30"
Else
    MsgBox "Failed to get time code editor!"
End If
```

Frames

Gets or sets the number of frames currently specified by the value currently in the time code editor control. This value depends on the frame rate used by the scene.

Sample Usage:

```
Dim timeCodeEditor
Set timeCodeEditor = ActiveScene.Control("Time Code Editor 1")

If Not(timeCodeEditor is Nothing) Then
    MsgBox "Current Frames: " + timeCodeEditor.Frames

    timeCodeEditor.Frames = 100
Else
    MsgBox "Failed to get time code editor!"
End If
```

Object

Each scene contains a collection of graphical and hardware elements called **Objects**, such as crawls and images, which are exposed to scripting using the Objects enumeration or Object function of a scene. Currently only object names are exposed, although additional functionality may be added in the future.

Name

Returns the name of this object.

Sample Usage:

```
Dim sceneObject
Set sceneObject = ActiveScene.Object("Text")

MsgBox sceneObject.Name ' Should display Text
```

Action

The Action can be used to interact with an animation of a scene. Actions can be accessed and used to trigger their underlying actions as follows:

Sample Usage:

```
Dim sceneAction
Set sceneAction = ActiveScene.Action("Hide")

sceneAction.Trigger 'Hide will be triggered
```

Name

Returns the name of this action, as apparent in the Prime Scene Builder at design time.

Sample Usage:

```
Dim sceneAction
Set sceneAction = ActiveScene.Action("Hide")

MsgBox sceneAction.Name ' Hide
```

Trigger

Triggers this action.

Sample Usage:

```
Dim action
Set sceneAction = ActiveScene.Action("FadeOff")

' Check if the action was returned
If Not(sceneAction is Nothing) Then
    ' Activate the animation
    sceneAction.Trigger
Else
    MsgBox "Failed to get an action!"
End If
```

Parameters

Scene authors can store and retrieve contextual information through the PRIME Scripting API by using either the top level PRIME project parameter collection (available as `Prime.Parameters`) or the parameter collection of a particular scene (available either through `ActiveScene.Parameters` or `Prime.Scene("1").Parameters`). The difference between the two styles is that the PRIME parameter collection persists as long as the application is running (or until the current project is changed), whereas the parameter collection of each scene is only persisted for as long as the scene is open. Once the scene is closed, all changes to the parameter collection are NOT persisted and upon reopening the scene, the parameters will have reverted to their initial state. The initial state of a scene parameter collection can be configured during the scene authoring process in the PRIME Scene Designer.

Both scene and project parameter collections can be accessed via scripting, which can retrieve or modify the values of existing parameters, add entirely new parameters or remove those that already exist.

Item(name As String)

Gets or sets the value of a parameter with the specified name.

Sample Usage:

```
' Display the current value of Name
MsgBox ActiveScene.Parameters.Item("Name")

' Change the value of Name
ActiveScene.Parameters.Item("Name") = "Jeff"
```

Items

Returns an enumeration of all parameters in the specified scene. An exception will be thrown if this property is accessed before the scene has been opened.

Sample Usage:

```
' Iterate through each parameter in the scene
For Each action in ActiveScene.Parameters.Items
        MsgBox parameter.Name & " = " & parameter.Value
Next
```

Contains(name As String)

Returns true if the parameters collection contains a parameter with the specified name.

Sample Usage:

```
If ActiveScene.Parameters.Contains("Name") Then
        MsgBox "Parameter exists!"
End If
```

Remove(name As String)

If the collection contains a parameter with the specified name, the parameter is removed.

Sample Usage:

```
ActiveScene.Parameters.Remove("Name")

If Not ActiveScene.Parameters.Contains("Name") Then
        MsgBox "Parameter no longer exists!"
End If
```

PRIME C# API Interaction

Although PRIME supports the original Channel Box VBScript API, there is additional integration with the C# API. **In Process** scripts now automatically have keywords defined for all objects within the executing scene. Each object provides access to as much of the C# API as possible, varying in levels of support from object type to object type. For example, accessing a Text Box control within the scene no longer requires use of the ActiveScene keyword. Instead the Text Box control may be accessed directly using its script friendly name:

```
' Supported, but no longer necessary!
' Dim textBox
' Set textBox = ActiveScene.Control("Text Box 1")
' textBox.Text = "ABC"

TextBox1.Text = "ABC"
```

The script-friendly name for an object follows the pattern below:

- Spaces are removed (e.g. "Text Box 1" becomes "TextBox1")
- Underscores are added (e.g. "123" becomes "_123")

Lua Script Auto Injections Using Tags

Tag Property Configuration

The Tag property enables auto injection by automatically populating script attributes with runtime information.

Injection Keywords

InjectChannelIndex

- Usage: Add `InjectChannelIndex` to the Tag property
- Requirement: Script must have a `ChannelIndex` attribute (Int32 type)

InjectParentAddress

- Usage: Add `InjectParentAddress` to the Tag property
- Requirement: Script must have a `ParentAddress` attribute (String type)

Tag Configuration Examples

Single Injection:

```
None
InjectChannelIndex
```

Multiple Injections:

```
None
InjectChannelIndex InjectParentAddress
```

Attributes Configuration

Required Declaration and Properties

- `Name:ChannelIndex`
  - `Type:Int32`
- `Name=ParentAddress`
  - `Type:String`

Accessing Attributes in Lua Script

None

```
-- Get injected values (automatically populated by Prime)
local channelIndex = Scene.get(Scene.this(), "ChannelIndex");
local parentAddress = Scene.get(Scene.this(), "ParentAddress");
```

Example 1: Basic Injection Usage

Tag Property

None

```
InjectChannelIndex InjectParentAddress
```

Lua Script - Retrieving Injected Values

None

```
-- Get the auto-injected values
local channelIndex = Scene.get(Scene.this(), "ChannelIndex");
local parentAddress = Scene.get(Scene.this(), "ParentAddress");

-- Display the injected information
Scene.logInfo("Running on Channel: " .. tostring(channelIndex));
```

```
Scene.logInfo("Parent Object Address: " .. parentAddress);
```

Example 2: Updating JavaScript Parameters

Show the example where Lua Script updates the **Parameter 1**(on **Java Script 1**)

Lua Tag Property

```
None
InjectChannelIndex InjectParentAddress
```

Script Declaration and Properties

- Name: ChannelIndex
  - Type: Int32
  - Value: 1
- Name: ParentAddress
  - Type: String
  - Value: “”
- Name: ScriptName
  - Type: String
  - Value: “**Java Script 1**”
- Name: ParameterName
  - Type: String
  - Value: “**Parameter 1**”

Lua Script - Parameter Update

```
None
-- Get injected runtime information
local channelIndex = Scene.get(Scene.this(), "ChannelIndex");
local parentAddress = Scene.get(Scene.this(), "ParentAddress");

-- Get custom configuration attributes
```

```
local scriptName = Scene.get(Scene.this(), "ScriptName");
local parameterName = Scene.get(Scene.this(), "ParameterName");

-- Create message with current channel and parent info
local message = string.format("Channel %d - Parent: %s",
channelIndex, parentAddress);

-- Send update using Client.send format
-- Format:
[channelIndex]\tUpdateParameter\t[parentAddress]/[scriptName]\t[p
arameterName]\t[value]
Client.send(channelIndex .. "\tUpdateParameter\t" ..
            parentAddress .. "/" ..
            scriptName .. "\t" ..
            parameterName .. "\t" ..
            message);
```

JavaScript Effect Utilization

The JavaScript effect can respond to parameter updates from Lua scripts:

JavaScript

```
// JavaScript Effect on Text Object
Parameter1.ValueChanged.connect(Parameter1Changed);

function Parameter1Changed()
{
    Parent.Text = "Received from Lua: " + Parameter1.Value;
}
```

When the Lua script sends an `UpdateParameter` message, the JavaScript(**Java Script 1**) parameter (**Parameter 1**) receives the new value and triggers the `ValueChanged` event,

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