



## OSMF Release Samples

### Walkthrough 10: Custom Plug-in Development

#### Overview:

This walkthrough provides some critical knowledge and steps for creating custom plug-ins. This walkthrough covers creating and implementing a custom, static plug-in that will apply a watermark, or bug, as a parallel element to all media being displayed and managed via the OSMF player.

The key class in creating an OSMF plug-in is the PluginInfo class. The PluginInfo class is the main class that interfaces with the components of the OSMF. The PluginInfo tells the OSMF what the plug-in can do, and provides the foundation, or base location, of where to start adding custom functionality.

The other key aspect of plug-in development with OSMF is understanding the types of plugins that can be built and the reasons for using each. There are two types of plugin types - proxy and reference. Proxy plugins return a ProxyElement that will be used to proxy another created MediaElement. Proxy plugins can non-invasively alter the behavior of the created MediaElement (for example, blocking the pause feature without changing the player application itself). A reference plugin depends on gaining reference to one or more MediaElements created by the MediaFactory (for example, a reference plugin that adds an image overlay that when clicked will pause the media playing).

In this walkthrough we will build a proxy plugin. The proxiedElement is the MediaElement of the content that was originally created in the OSMF player via the MediaFactory prior to the plug-in being involved. When a MediaElement is created via the MediaFactory the proxiedElement property is set on any proxy plugins that can handle that resource. Usually the proxy plugin will then generate a new MediaElement to that the plugin will then work with.

The idea with plug-ins is that in the PluginInfo constructor, the list of supported MediaFactoryItem's and the method to call once the element has been completed should be passed to the super. That list of MediaFactoryItem's is then used by the MediaFactory in the main player to assess whether-or-not the plug-in needs to handle any of the media elements that are being generated. Each MediaFactoryItem defines what type of content it is able to handle, as well as a method pointer to generate the appropriate MediaElement. After the MediaFactory has generated the proper element for the content the player is attempting to play, the MediaFactory references the list of MediaFactoryItem's from the loaded plug-in. If a match is found the appropriate media element creation function in the plug-in is called. Once that custom function returns a MediaElement, the proxiedElement property is set on the new plug-in-generated MediaElement. The proxiedElement is the original MediaElement generated by the MediaFactory, and can then be modified by the plug-in. The possibly

modified, or custom MediaElement from the plug-in is what eventually gets passed along to be displayed and controlled via the OSMF player.

In this walkthrough we will use the above sequence, and the setting of the proxiedElement, to intercept the original MediaElement that is to be played, and instead pass along a ParallelElement with a watermark.

**NOTE:** The id used for the MediaFactoryItem will impact the priority of which MediaFactoryItem is used to generate the appropriate MediaElement if more than 1 MediaFactoryItem matches the criteria to generate the element. Any id that doesn't include 'org.osmf' will be given priority. This is valuable as MediaFactoryItem's are added dynamically, and may duplicate generic ones within the MediaFactory - and thus this gives priority to the custom MediaFactoryItem's.

Creating a custom SWF-based dynamic plug-in is not covered in this walkthrough. However, to generate a dynamic plug-in, all that is needed is to create a new custom class that extends MovieClip. Within that custom class, implement a getter method for the property pluginInfo(), and return an instance of your custom PluginInfo class. Compile the custom Sprite class, and there you have a dynamic plug-in.

## Objectives:

- Understand the building blocks of a custom plug-in:
  - PluginInfoResource
  - PluginInfo Object
  - Creating & Managing a ProxyElement
- Build a custom plug-in that does the following:
  - Applies a watermark/bug to all media being displayed - positioned appropriately
  - Integrate in AS3 traces and ExternalInterface based JavaScript communication for info and debug info during playback
- Load and test the custom plug-in as a static plug-in

## Setup

1. Open the file WT10\_CustomPluginDevelopment.as in the {SAMPLES\_PROJECT}/src directory.
2. Set the class file as the application file to compile. There are two different ways of doing this depending on which program you are building your application in.
  - Flash Builder**  
Right-click the WT10\_CustomPluginDevelopment.as file and select Set as Default Application from the context menu that appears. This will add the project to the list of compilable applications. A blue dot on the file icon indicates that the file is the default application file.
  - Flash Professional**  
Open the OSMF\_SampleTemplate.fla and save it as WT10\_CustomPluginDevelopment.fla. Then change the document class for the file (in the Properties panel) to WT10\_CustomPluginDevelopment.
3. Open CustomSamplePluginInfo.as in the {SAMPLES\_PROJECT}/com/realeyes/plugins/custom/ directory.

4. Open CustomSampleElement.as in the {SAMPLES\_PROJECT}/com/realeyes/plugins/custom/element/ directory.

NOTE: These files have been provided as a starting point for these walkthroughs.

## Loading the plug-in

5. In WT10\_CustomPluginDevelopment.as in the default package locate the comment that begins "//Marker 1:".
6. Create a new PluginInfoResource variable named pluginResource.
7. Pass a new CustomSamplePluginInfo object as the only parameter to the constructor.

```
//Marker 1: Create the plugin resource using a static plugin  
var pluginResource:MediaResourceBase = new PluginInfoResource( new  
CustomSamplePluginInfo() );
```

## Building the PluginInfo object

8. Open the CustomSamplePluginInfo class in the {SAMPLES\_PROJECT}/com/realeyes/plugins/custom/ directory.
9. Under the "//Marker 2:" comment in the constructor create a Vector variable named items - the Vector type should be MediaFactoryItem and set it equal to a new MediaFactoryItem Vector.

```
//Marker 2: Specify the media items that this plugin will handle  
var items:Vector.<MediaFactoryItem> = new Vector.<MediaFactoryItem>();
```

10. Under the comment "//Marker 3:" create a NetLoader variable named loader and set it equal to a new NetLoader.

```
//Marker 3: Let the MediaFactory know what kinda of media the plugin  
can handle...  
var loader:NetLoader = new NetLoader();
```

11. Create a new MediaFactoryItem variable named item.
12. Set item equal to a new MediaFactoryItem object.
13. The first constructor parameter should be the string 'com.realeyes.plugins.custom.sample'.
14. The second parameter is the canHandleResource on the loader object.
15. The third parameter is the function createMediaElement
16. The fourth and final parameter is MediaFactoryItemType.PROXY.

```

//Marker 3: Let the MediaFactory know what kinda of media the plugin
can handle...
var loader:NetLoader = new NetLoader();
var item:MediaFactoryItem = new MediaFactoryItem(
    "com.realeyes.plugins.custom.sample",
    loader.canHandleResource,
    createMediaElement,
    MediaFactoryItemType.PROXY
);

```

**NOTE:** This MediaFactoryItem will handle anything a NetLoader can handle and will call the createMediaElement method of the CustomSamplePluginInfo class. Once it returns the MediaElement, it will set the proxiedElement property to the original MediaElement generated by the MediaFactory in the player.

17. Add the item to the items array by calling the push() method, passing it the item MediaFactoryItem.

```

//Marker 3: Let the MediaFactory know what kinda of media the plugin
can handle...
var loader:NetLoader = new NetLoader();
var item:MediaFactoryItem = new MediaFactoryItem(
    "com.realeyes.plugins.custom.sample",
    loader.canHandleResource,
    createMediaElement,
    MediaFactoryItemType.PROXY
);
items.push( item );

```

18. In the createMediaElement() method, under the "//Marker 4:" comment, return a new CustomSampleElement.

```

//Marker 4: Create a new CustomSampleElement
return new CustomSampleElement();

```

## Building the ProxyElement

19. Open CustomSampleElement.as in the {SAMPLES\_PROJECT}/com/realeyes/plugins/custom/element/ directory. Note that the constructor doesn't do anything other than pass the proxiedElement parameter to the super. At this point in our plug-in, the proxiedElement has not been set, and the parameter of the constructor is null.

What really does all the work and sets up the MediaElement we want to use is the setter of the proxiedElement, since it will not be called or set until after the class has been instantiated.

20. in the setter for the proxiedElement property, under the "//Marker 5:" comment, create an ImageElement variable named image.
21. Set image equal to new ImageElement, passing a new URLResource as the only parameter to the constructor, using the string 'assets/osmf\_stacked.png'.

```
//Marker 5: Create the image element
var image:ImageElement = new ImageElement( new URLResource( "assets/
osmf_stacked.png" ) );
```

22. Create a new LayoutMetadata variable named layout, and set it equal to a new LayoutMetadata object;

23. Set the following properties and values on the layout object:

1. width = 75
2. height = 66
3. bottom = 0
4. right = 0

```
//Marker 5: Create the image element
var image:ImageElement = new ImageElement( new URLResource(
"assets/osmf_stacked.png" ) );
var layout:LayoutMetadata = new LayoutMetadata();
layout.width = 75;
layout.height = 66;
layout.bottom = 0;
layout.right = 0;
```

24. Add the layout LayoutMetadata to the image ItemElement

```
//Marker 5: Create the image element
var image:ImageElement = new ImageElement( new URLResource( "assets/
osmf_stacked.png" ) );
var layout:LayoutMetadata = new LayoutMetadata();
layout.width = 75;
layout.height = 66;
layout.bottom = 0;
layout.right = 0;
image.addMetadata( LayoutMetadata.LAYOUT_NAMESPACE, layout );
```

25. Under the "//Marker 6:" comment, create a new ParallelElement named parallel.

```
//Marker 6: create a parallel element so we can add the image &
original element
var parallel:ParallelElement = new ParallelElement();
```

26. Under "//Marker 7:", add 'value' and 'image' as children of the parallel element.

```
//Marker 7: Add the proxied element and the image as children
parallel.addChild( value );
parallel.addChild( image );
```

27. Under the "//Marker 8:" comment, set the super.proxiedElement equal to the parallel element.

```
//Marker 8: Set the proxied element
super.proxiedElement = parallel;
```

28. Save and run the application. The video should play, and the OSMF logo will be placed on top of the video in the bottom right corner.



29. Return to the CustomSampleElement class.
30. Under the "//Marker 9:" comment, add an event listener for the MediaElementEvent.TRAIT\_ADD and set the event handler method to be \_onAddTrait.  

```
//Marker 9: Add trait add and remove listeners
proxiedElement.addEventListener( MediaElementEvent.TRAIT_ADD,
_onAddTrait );
```
31. Add an event listener for MediaElementEvent.TRAIT\_REMOVE to be handled by the \_onRemoveTrait method.  

```
//Marker 9: Add trait add and remove listeners
proxiedElement.addEventListener( MediaElementEvent.TRAIT_ADD,
_onAddTrait );
proxiedElement.addEventListener( MediaElementEvent.TRAIT_REMOVE,
_onRemoveTrait );
```
32. The \_onRemoveTrait() event handler method traces out the trait type that is being removed.
33. In the \_onAddTrait() event handler method, we will need to evaluate which trait is being added, and react to that trait. In this method, a switch statement has been created that evaluates and traces the 'traitType' property of the

MediaElementEvent. Locate the "//Marker 10:" comment.

34. Under the "//Marker 10:" comment, create a new TimeTrait variable named timeTrait.
35. Set timeTrait equal to the result of calling the getTrait() method on the proxiedElement. Pass the trait type of MediaTraitType.TIME to the getTrait() method as the only parameter, making sure to cast the result as a TimeTrait object.

```
//Marker 10: Get the time trait, so we can handle the duration changed event
var timeTrait:TimeTrait = proxiedElement.getTrait( MediaTraitType.TIME
) as TimeTrait;
```

36. Add an event listener to the timeTrait object for the TimeTrait.DURATION\_CHANGE event. Handle this event with the \_onDurationChanged() method. The \_onDurationChanged() method traces the value of the changed duration to the debugger.

```
//Marker 10: Get the time trait, so we can handle the duration changed event
var timeTrait:TimeTrait = proxiedElement.getTrait( MediaTraitType.TIME
) as TimeTrait;
timeTrait.addEventListener( TimeEvent.DURATION_CHANGE,
_onDurationChanged );
```

37. Save all of the files and run the application in debug mode. you should see debugger output similar to what's shown in the image below. The output to note would be the 'New Duration:'. This is what is being output when the event handler for TimeEvent.DURATION\_CHANGE is executed.

The screenshot shows a web browser window displaying a video player. The video player has a dark background with the title "A Faery's Tale" in a stylized, glowing orange font. In the bottom right corner of the video player, there is a logo for "open source media framework".

Below the video player, the browser's address bar shows the URL "http://localhost/osmf\_samples/WT10\_CustomPluginDevelopment...". The browser's toolbar includes icons for "Console", "Data/Services", "Network Monitor", and "Progress".

The console window is open, showing the following log messages:

```
WT10_CustomPluginDevelopment [Web Application] http://localhost/osmf_samples/WT10_CustomPluginDevelopment
loadMedia()
Create plugin element
Element created!
Media has a timeline
[SWF] /osmf_samples/WT10_CustomPluginDevelopment.swf - 593,406 bytes after decompression
Media is playable
New Duration: 31.671
Media is seekable
```