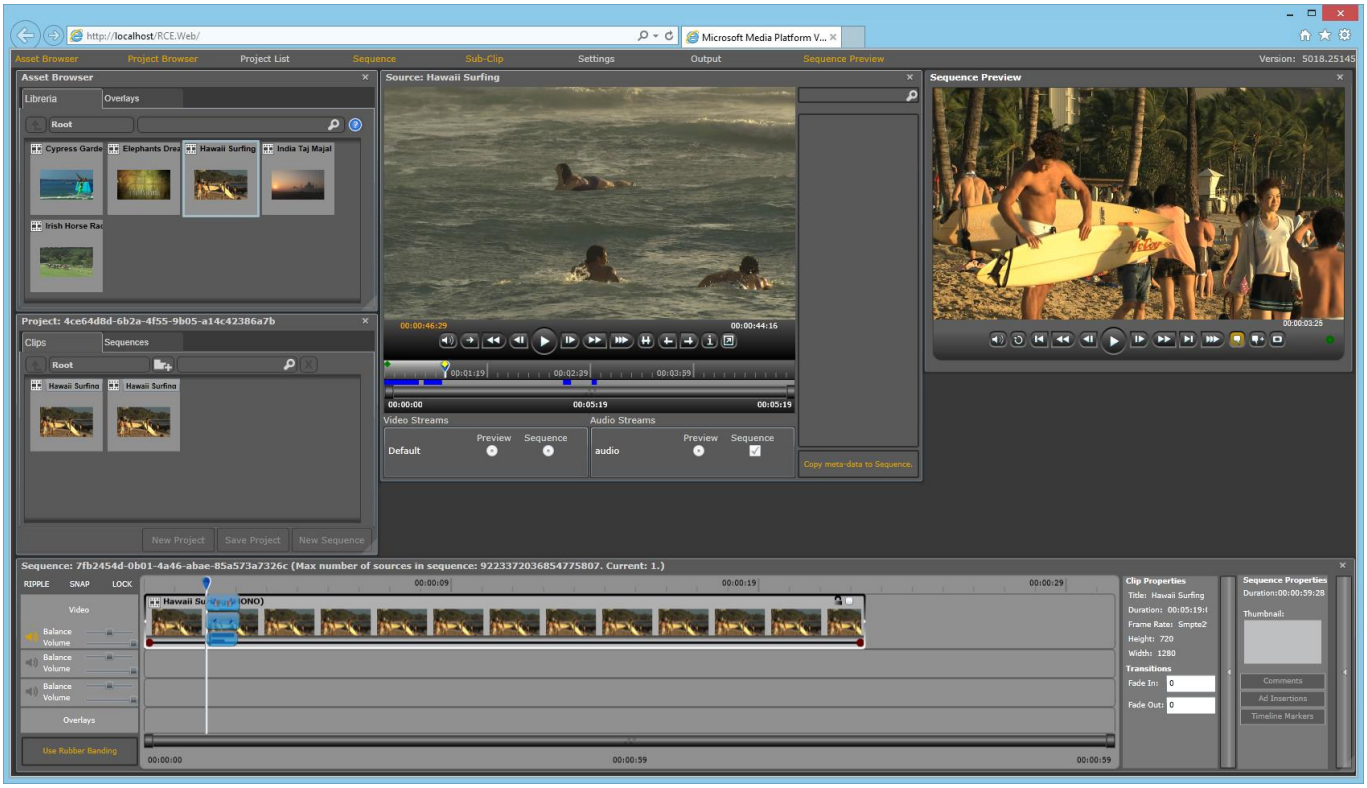
Microsoft Media Platform Video Editor 2.1

Contents

Overview

The Silverlight Microsoft Media Platform Video Editor (MMP Video Editor) simplifies the editing and publishing process, enabling real-time, browser-based video editing and providing the ability for publishers to improve collaboration, manage dynamic metadata and deliver exciting content to the Web.

MMP Video Editor provides media organizations with a lightweight, Web-based tool that editors anywhere can use to assemble and edit video, audio, images, and XAML overlays with time code accurate control. Powered by Microsoft Silverlight, the MMP Video Editor provides a rich, stutter-free, full-screen editing experience. The MMP Video Editor has an open and extensible architecture, and it can be integrated into any digital asset management (DAM) and any encoding/transcoding system that an organization is already using.



Microsoft Media Platform Video Editor

Below are a few of the benefits offered by Silverlight MMP Video Editor solution from Microsoft:

- Streamlines the movement of audio and video content from the source to any distribution platform including the Web
- Keeps costs down - MMP Video Editor is a free download that runs in a browser, so any organization in need of a rough cut editing tool can quickly and cost-effectively deploy it to the individuals in the organization who need it
- Provides the ability for editors to interact with the MMP Video Editor from anywhere through browsers running on any PC or Macintosh system
- Enables editors to move long form video assets through Web-based editors and publish to Web sites within seconds

New Features in 2.1 Release

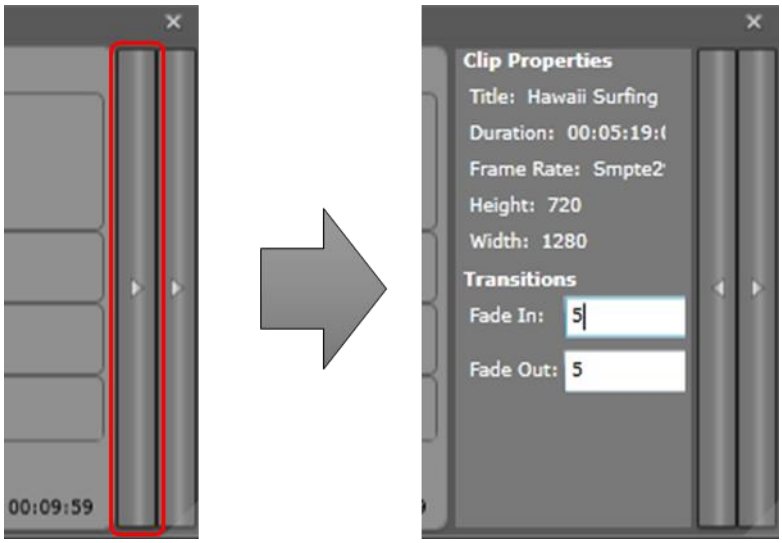
This document provides an overview of the new features included in this new version of the Microsoft Media Platform (MMP) Video Editor (formerly known as RCE).

The features covered in this document are:

- Upgraded Visual Studio 2012 solution
- Support for fade in and fade out transitions
- Support to load asset in XML format from a JavaScript bridge so you can query your assets directly from a CMS web page and send them back to the MMP Video Editor
- New mode to snap mark in and mark out positions to the nearest fragment boundary

Fade In and Fade Out Transitions

The MMP Video Editor 2.1 allows editors to add fade in and fade out transitions for the clips. Once a clip is added to the Sequence Timeline, its properties can be modified in the properties panel. To visualize the properties panel, click the first expandable arrow at the right of the timeline. Make sure to select the clip in the Sequence Timeline to which you want to apply the transition, and then configure the **Fade In** and **Fade Out** properties accordingly (in seconds).



Configuring Clip's Transitions

Sequence Timeline Clip with In / Out Transitions

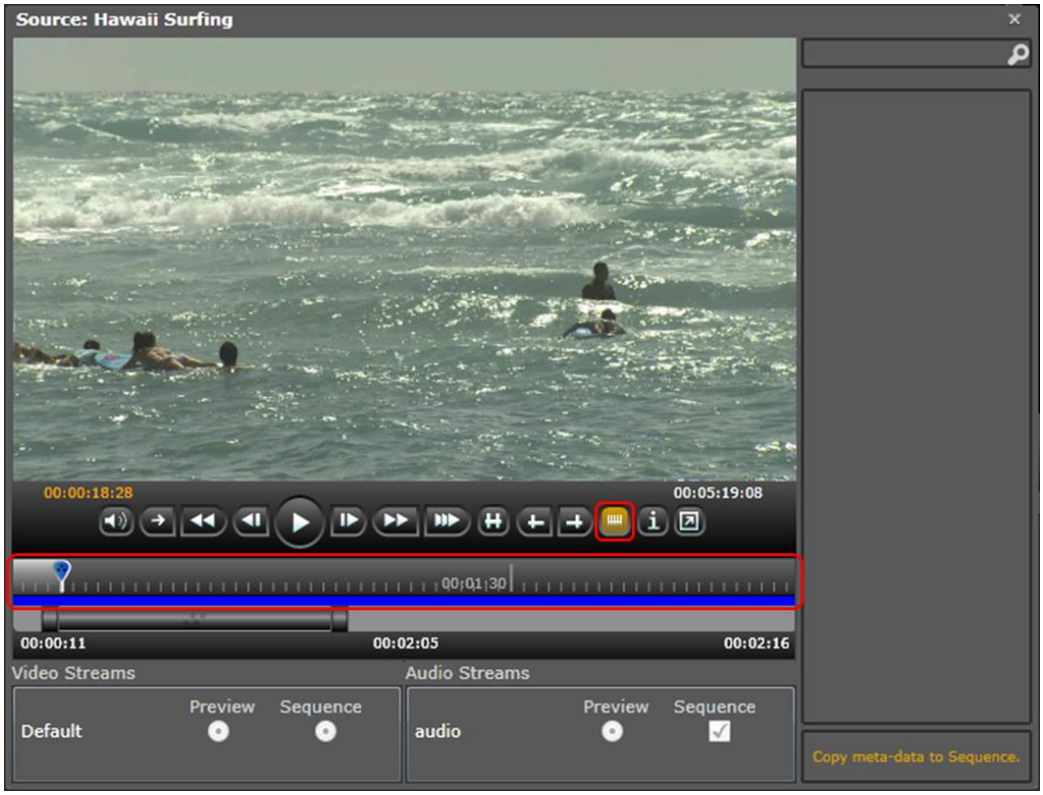
Load Assets from JavaScript Bridge

The MMP Video Editor 2.1 provides a new way to populate the asset library, by using a JavaScript bridge with a CMS Web page. For more details, refer to SCENARIO C: I want to load content to the MMP Video Editor library from a Web page in my CMS.

Snap to Fragments Boundaries Mode

The MMP Video Editor 2.1 provides a new mode to make it easier for editors to perform cuts at fragment boundaries. This functionality is available when adding mark ins and mark outs to both the Sub-Clip Tool and Sequence windows:

- **Sub-Clip Tool window**.
    - A new button appears in the Sub-Clip Tool window to let the editor see fragment boundary marks in the timeline.
    - When the editor adds a mark in, the mark in position is snapped to the **start** position of the current fragment. If the video is paused, then the play head position in the timeline is moved to the new mark in position.
    - When the editor adds a mark out, the mark out position is snapped to the **end** position of the current fragment. If the video is paused, then the play head position in the timeline is moved to the new mark out position.
    - When the editor seeks in the timeline, the play head position is snapped to the nearest fragment boundary (begin or end).
- **Sequence window**.
    - When the editor adjusts the clip size in the timeline (from both the beginning and the end), the mark in/out position is snapped to the nearest fragment boundary.



Snap to Fragments Boundaries Mode

To enable the *Snap to Fragments Boundaries* mode, change the value of the **SnapToFragmentBoundaries** setting to **true** in the **Settings.xml** file located in the **RCE.Web** project.

XML

```
<Parameter Name="SnapToFragmentBoundaries" Value="true"/>
```

Deployment Guide

The Microsoft Media Platform Video Editor is a standalone component proxy and metadata editing solution that allows end users to collaborate on and edit audiovisual content in a cross platform web browser experience.

The MMP Video Editor was built using the Microsoft Composite Application Guidance for WPF and Silverlight (a.k.a. Prism) developed by Microsoft Patterns & Practices. Prism includes guidance and a library, which are designed to help manage the complexities faced when building enterprise level WPF client applications and RIAs with Silverlight. Prism helps design and build flexible client applications using loosely coupled, independently evolvable pieces that work together and are integrated into the application, also known as composite applications.

The following are the pre-requisites for the installation of the MMP Video Editor on the application server:

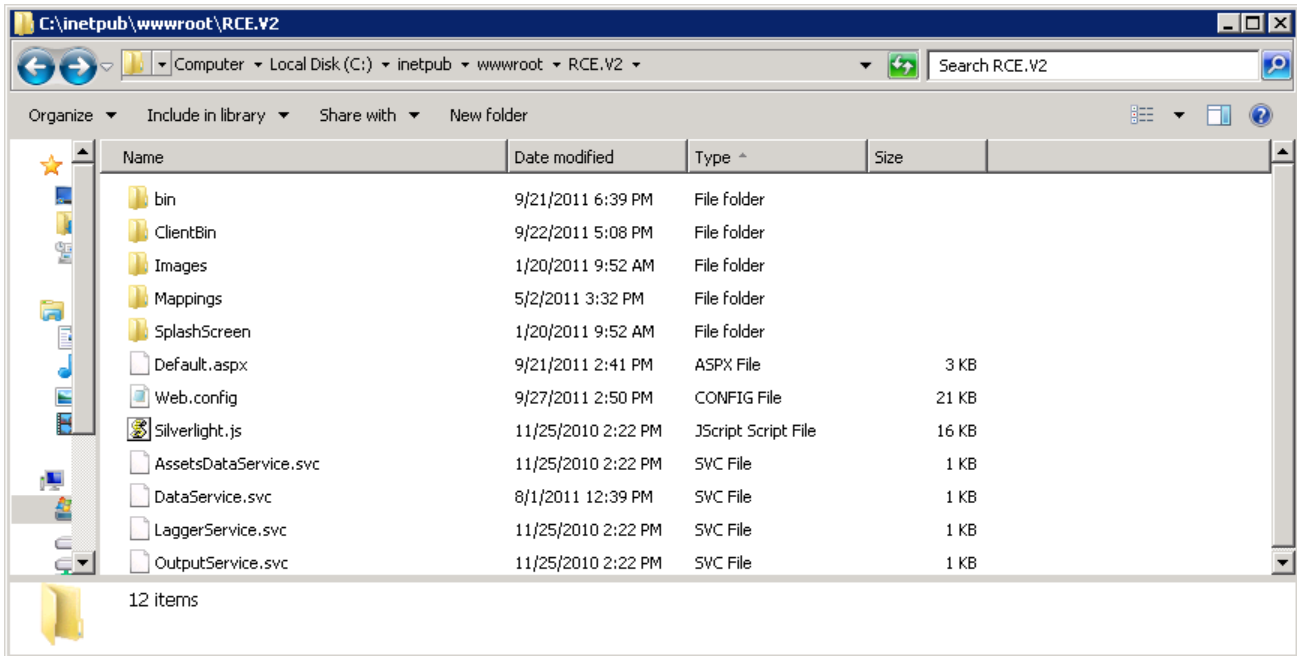
- Visual Studio 2012 Update 3
- NuGet Package Manager 2.7 or higher
- .NET Framework 4.0
- Silverlight 5
- Silverlight 5 Toolkit
- Windows Communication Foundation HTTP and non-HTTP activation
- IIS 6.0 or higher with ASP.NET enabled

Once the aforementioned prerequisites are installed, the MMP Video Editor can be deployed.

Configuring Internet Information Services

1. Build the entire **RoughCutEditor** solution. To do this, open the **RoughCutEditor** solution in Visual Studio 2012 as Administrator, and press **F6**.
   Important: You may need to force the build twice to restore all the NuGet packages required. If you keep getting the following build error message, try closing the **RoughCutEditor** solution and opening it again.
   ```
   The build restored NuGet packages. Build the project again to include these packages in the build. For more information, see http://go.microsoft.com/fwlink/?LinkID=317568.
   ```

2. The **RCE.Web** project is configured to create the *RCE.Web* virtual directory in your local IIS. If you want to deploy it to a different frontend server, copy the contents of the **RCE.Web** directory to the directory you want to deploy the application.



RCE.Web directory

3. Launch **Internet Information Services (IIS) Manager**.
4. On IIS, create a new **Virtual Directory**, pointing to the directory selected in Step 1.
5. Validate that you have the required mime type to handle Silverlight packages. In order to verify this:
    a. Select the **MMP Video Editor** virtual directory, and click MIME Types… button.
    b. If the .xap extension is not registered, then click **New...**
    c. Fill the extension and mime types fields with the following values:
        i. **Extension**: .xap
        ii. **Mime** Type: application/x-silverlight-app
    d. Click **OK** to register the mime type.

Configuring the MMP Video Editor

Having copied the MMP Video Editor application to the virtual directory, the application can be configured.

Configuring the Settings

1. Open the Windows Explorer and go to the directory where the MMP Video Editor is located.
2. Open the **RCE.Web** folder.
3. Open the **Settings.xml** file.

```
<?xml version="1.0" encoding="utf-8" ?>
<Settings>
  <Parameter Name="ModulesCatalog" Value="ModulesCatalog.xaml"/>
  <Parameter Name="DataServiceUrl" Value="http://localhost/RCE.Web/DataService.svc"/>
  <Parameter Name="AssetsDataServiceUrl" Value="http://localhost/RCE.Web/AssetsDataService.svc"/>
  <Parameter Name="LoggerServiceUri" Value="http://localhost/RCE.Web/LaggerService.svc"/>
  <Parameter Name="OutputServiceUrl" Value="http://localhost/RCE.Web/OutputService.svc"/>
  <Parameter Name="UserName" Value="RCE\Default" />
  <Parameter Name="CDN" Value="http://rcecdn" />
  <Parameter Name="MetadataFields" Value="Title;Duration;Frame Rate;Height;Width" />
  <Parameter Name="MaxNumberOfItems" Value="1000" />
  <Parameter Name="CommentTypes" Value="Global;Playhead;Shot;Ink" />
  <Parameter Name="UndoLevel" Value="20" />
  <Parameter Name="SearchWithinAssetsLimit" Value="15"/>
  <Parameter Name="MaxNumberOfAudioTracks" Value="2"/>
  <Parameter Name="SnapModeEnabled" Value="false"/>
  <Parameter Name="RippleModeEnabled" Value="false"/>
  <Parameter Name="DefaultFrameRate" Value="Smpte2997NonDrop"/>
  <Parameter Name="DefaultTimelineDurationInSeconds" Value="60"/>
  <Parameter Name="DefaultThumbnailImageUri" Value="http://localhost/RCE.Web/Images/DefaultThumbnail.jpg"/>
  <Parameter Name="CommentDurationInSeconds" Value="60"/>
  <Parameter Name="MinBitrate" Value="200000"/>
  <Parameter Name="MaxBitrate" Value="10000000"/>
  <Parameter Name="SingleBitrate" Value="true"/>
  <Parameter Name="KeyboardMappings" Value="RCE|http://localhost/RCE.Web/Mappings/RCEKeyboardMappings.xml;AVID|http://localhost/RCE.Web/Mappings/AVIDKeyboardMappings.xml;" />
  <Parameter Name="EnableChannelsConvention" Value="false"/>
  <Parameter Name="GapVideoUrl" Value="http://devplatem.vo.msecnd.net/RCEAzureDemo/gapvideo/gap.ism/manifest" />
  <Parameter Name="GapCMSId" Value="1234567890" />
  <Parameter Name="GapAzureId" Value="0987654321" />
  <Parameter Name="SearchIntegrationEnabled" Value="false" />
  <Parameter Name="SearchPopupUrl" Value="http://localhost/RCE.Web/popup.html"/>
  <Parameter Name="SearchPopupWidth" Value="800"/>
  <Parameter Name="SearchPopupHeight" Value="200"/>
  <Parameter Name="SnapToFragmentBoundaries" Value="false"/>
  <Parameter Name="LogEntryElementName" Value="InStreamEnvelope"/>
  <Parameter Name="LogDataElementName" Value="PlayByPlay"/>
  <Parameter Name="LogEntryPollingIntervalInSeconds" Value="60"/>
  <Parameter Name="LogEntrySourceUrl" Value="http://localhost/RCE.Web/media/logentries.xml"/>
  <!--supported modes are embedded | polling. in the future, web service will be available-->
  <Parameter Name="LogEntriesRetrievalMode" Value="embedded"/>
  <Parameter Name="TreatGapAsError" Value="false"/>
</Settings>
```

Settings.xml File

4. Replace the value of the following settings:
    a. **DataServiceUrl**. Replace the default hostname with the hostname where the DataService.svc service was published.
    b. **AssetsDataServiceUrl**. Replace the default hostname with the hostname where the AssetsDataService.svc was published.
    c. **LoggerServiceUri**. Replace the default hostname with the hostname where the LaggerService.svc was published.
    d. **OutputServiceUrl**. Replace the default hostname with the hostname where the OutputService.svc was published.
    e. **KeyboardMappings**. Replace the default hostname with the hostname where the keyboard mappings folder was published.
    f. **GapVideoUrl**. Replace with the URL of the video to be used to fill gaps in the CSM. Take into account that the audio encoding settings for different videos must not differ within a single CSM.
    g. **DefaultThumbnailUri**. Replace the default hostname with the hostname where the RCE was published. The Default Thumbnail URI is used when assets do not provide a thumbnail of their own.
5. (Optional) Below is an explanation of other options that can be found and customized in the settings file.
    a. **ModulesCatalog**. The name of the Module Catalog file that defines the modules being loaded. Note: The Module Catalog is where all the application modules are defined.
    b. **UserName**. Defines the username used by the application.
    c. **CDN**. URL where the content was published in the data store. If the asset URL does not contain a valid URL, then this value will be appended to the asset URL. For example: http://{servername}/Media
    d. **MediaLibraryUri**. URL of the container to retrieve in the Media Library of the MMP Video Editor.
    e. **MetadataFields**. Defines the metadata fields being shown for the different assets available.

f. **MaxNumberOfItems**. Defines the max number of items (assets) being retrieved from the server. The default value is 1000.
g. **CommentTypes**. Defines the comment types available in the MMP Video Editor. By default all the comment types are available (*Global*, *Playhead*, *Shot*, *Ink*).
h. **UndoLevel**. Defines how many undo/redo operations will be saved. The default value is 20.
i. **SearchWithinAssetsLimit**. Defines the max number of items being retrieved from the PBP or Commentary metadata of a smooth streaming asset.
j. **MarkInOffsetInSeconds**. Defines the offset to apply to the Mark In (start position) value of a Play-By-Play or Commentary item.
k. **MaxNumberOfAudioTracks**. Defines the maximum number of audio tracks allowed in the Microsoft Media Platform Video Editor.
l. **SnapModeEnabled**. Defines whether or not the snap mode is enabled. When the snap mode is enabled, any asset dropped to the timeline will be positioned just after the previous asset.
m. **SnapToFragmentBoundaries**. Defines whether or not the mark in and mark out positions are snapped to the nearest fragment boundary. When the snap to fragment boundaries is enabled, all the mark in and mark out positions are automatically adjusted to the nearest chunk.
n. **SearchIntegrationEnabled**. Defines if the assets are loaded from a JavaScript bridge with a CMS or from the Asset Data Service.
o. **SearchPopupUrl**. If **SearchIntegrationEnabled** is enabled, this URL represents the Web page that will be open to search for assets in the JavaScript bridge.
p. **DefaultTimelineDurationInSeconds**. Defines the default duration of the timeline in seconds.
q. **MinBitrate**. Defines the minimum quality bitrate to be used for playback.
r. **MaxBitrate**. Defines the maximum quality bitrate to be used for playback.
s. **SingleBitrate**. Defines if only a single bitrate should be used for playback. Useful for taking advantage of the cache.
t. **EnableChannelsConvention**. If enabled, channel one and two balance audio to different sides to provide a stereo experience.

6. Save the **Settings.xml** and close it.
7. Open the **Default.aspx** file.
8. (optional) Update the **initParams** value by replacing the default hostname with the hostname where the **Settings.xml** file was published. By default, the MMP Video Editor will look for the settings file in the same hostname where the **Default.aspx** file is hosted.

Configuring the Data Providers and Metadata Strategies

1. Open the Windows Explorer and go to the directory where the MMP Video Editor is located.
2. Open the **RCE.Web** folder.
3. Locate the **<unity>** section within the **Web.config** file. Under the **<containers>/<container>/<types>** subsection, the available MMP Video Editor data providers, asset data providers and metadata strategies are listed.

```
<unity>
  <typeAliases>...</typeAliases>
  <containers>
    <container>
      <types>
        <!-- Logger -->
        <type type="LoggerService" mapTo="LAgger.LoggerSe">...</type>

        <!-- Data Providers -->
        <!-- Demo Data Provider -->
        <type type="RCE.Services.Contracts.IDataProvider, RCE.Services.Contracts" mapTo="RCE.Data.Demo.DemoDataProvider, RCE.Data.Demo">
          <lifetime type="external" />
        </type>
        <!-- SQL Data Provider -->
        ...

        <!-- Assets Data Providers -->
        <!-- Xml Assets Data Provider -->
        <type type="RCE.Services.Contracts.IAssetsDataProvider, RCE.Services.Contracts" mapTo="RCE.Data.Xml.XmlAssetsDataProvider, RCE.Data.Xml">
          <lifetime type="external" />
        </type>
        <!-- SQL Assets Data Provider -->
        ...

        <!-- Demo Assets Data Provider -->
        ...
        ...
        <!-- FileSystem Assets Data Provider -->
        ...
        <!-- Metadata Locator -->
        <type type="RCE.Services.Co" mapTo="RCE.Services.Me">...</type>
        <!-- Metadata Strategies -->
        <type type="RCE.Services.Co" mapTo="RCE.Metadata.St" name="WMMetadataStrat">...</type>
        <type type="RCE.Services.Co" mapTo="RCE.Metadata.St" name="ImageMetadataSt">...</type>
        <type type="RCE.Services.Co" mapTo="RCE.Metadata.St" name="SmoothStreaming">...</type>
      </types>
    </container>
  </containers>
</unity>
```

Web.config File

4. Check that the data provider and the asset data provider are properly configured. In order to start using the MMP Video Editor, both DataProvider and AssetsDataProvider must be defined.
5. To retrieve metadata associated with the assets, check that the metadata strategies are configured.
6. Save the **Web.config** file and close it.

Running the MMP Video Editor

At this point, the MMP Video Editor should be properly configured and deployed. To run the MMP Video Editor, navigate to the URL where you published the application. It should be similar to **http://*{hostname}*/MMPVE/Default.aspx**.

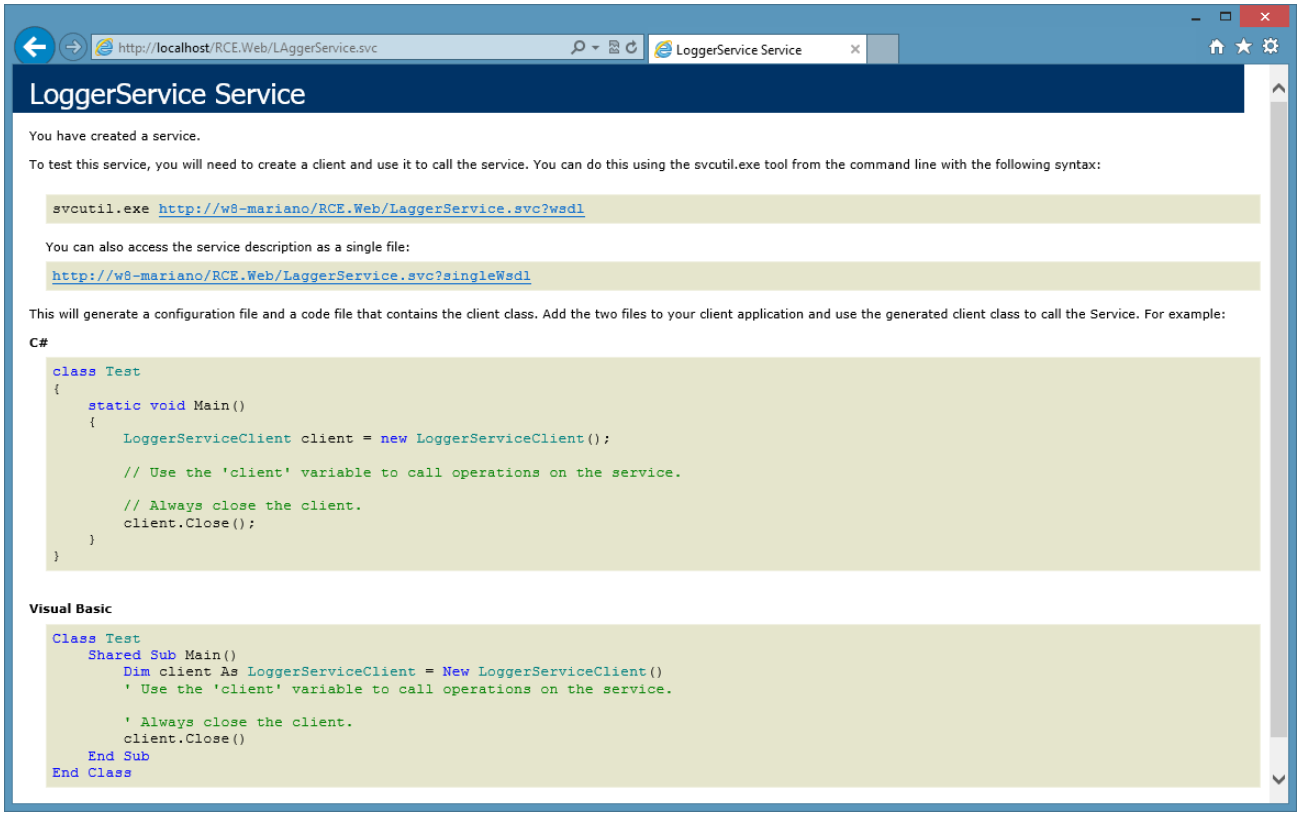
Note: You must have a license to parse H264 CodePrivateData in order to use the MMP Video Editor.

Troubleshooting Guide

The MMP Video Editor is opened but the Library is empty. The browser reports JavaScript errors.

This issue is likely related to WCF configuration. In the following steps, you are going to verify if WCF is correctly configured.

1. Open the browser.
2. Navigate to http://localhost/RCE.Web/LAggerService.svc
3. If WCF is correctly configured, the following screen will appear:



WCF correctly configured

4. If the screen that appears does not resemble that of the previous image, that might indicate that WCF is not correctly configured. If the error returned is a "HTTP Error 404 - Not Found", this might be due to an issue with the WCF scriptmaps registrations. To solve this issue:
    a. Open a command line console as Administrator
    b. Browse to **%windir%\Microsoft.NET\Framework\v3.0\Windows Communication Foundation**
    c. Run the command **ServiceModelReg.exe –i**

The playback of the videos on the library is not working.

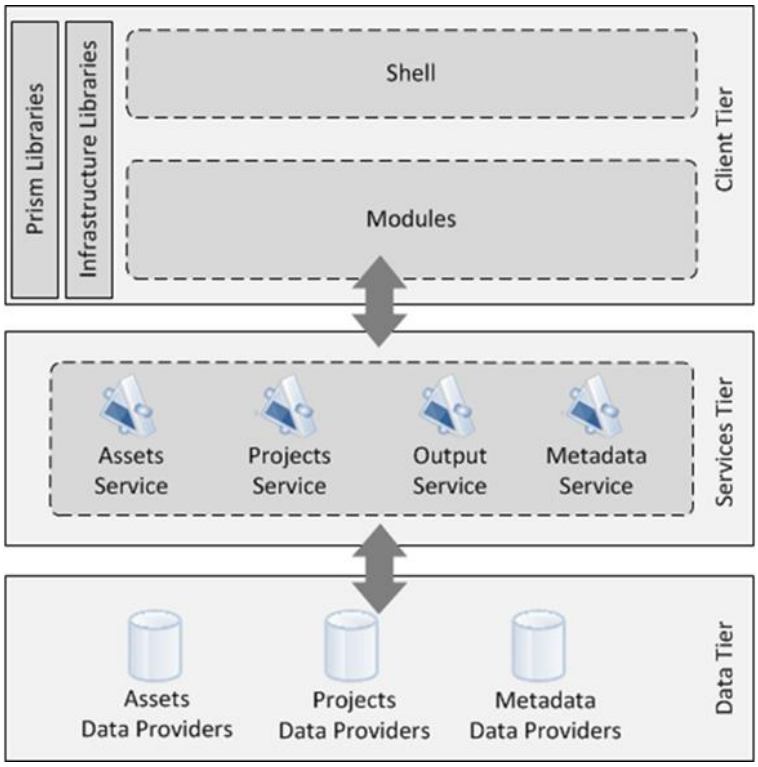
The default instance of the MMP Video Editor is consuming videos from an external server. If the playback of the videos is not working as expected, it might be due to an unexpected downtime of the external server.

High Level Architecture

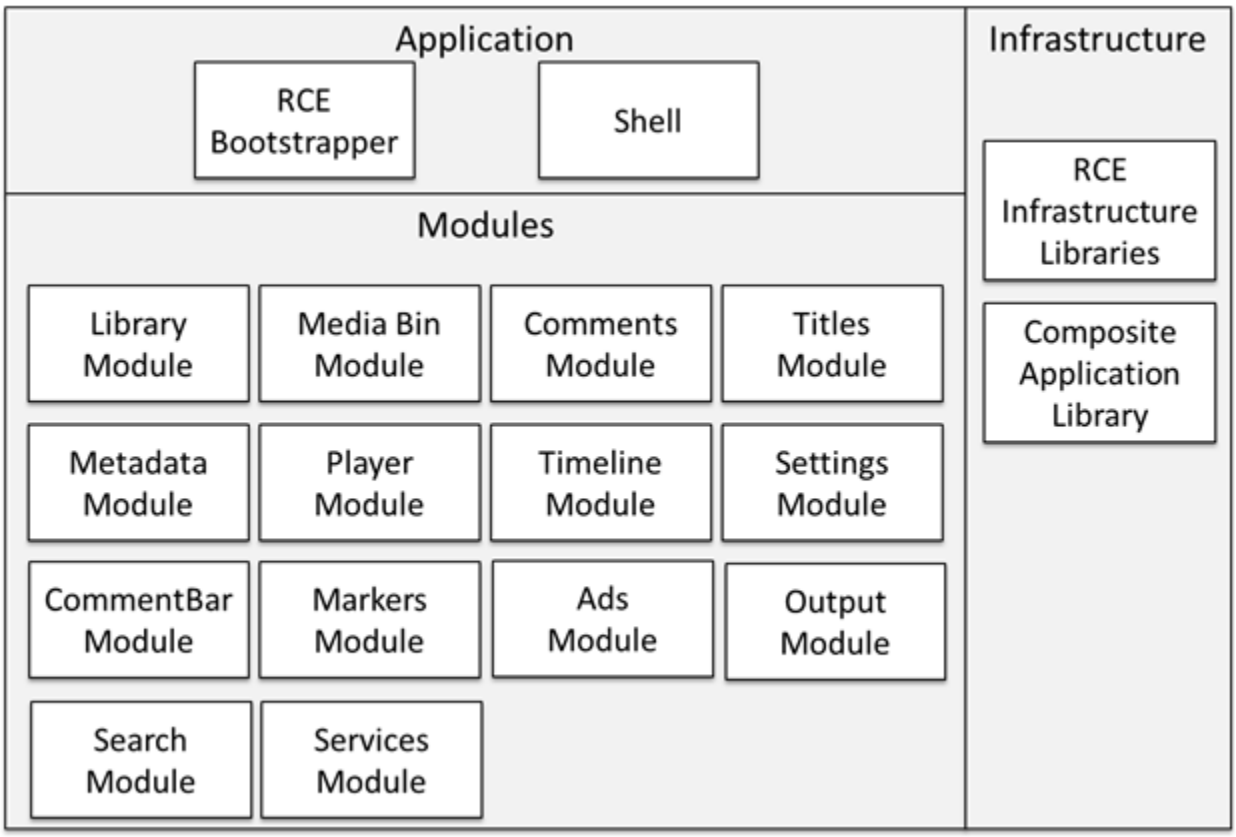
The architecture on the MMP Video Editor is divided in three tiers:

1. The client or presentation tier which displays information the user can understand.
2. The services tier which coordinates the application logic. It also moves and processes data between the two surrounding layers.
3. The data tier which manages how the information is being stored and retrieved.

The following diagram represents the high-level architecture of the Microsoft Media Platform Video Editor.



High Level Architecture Diagram



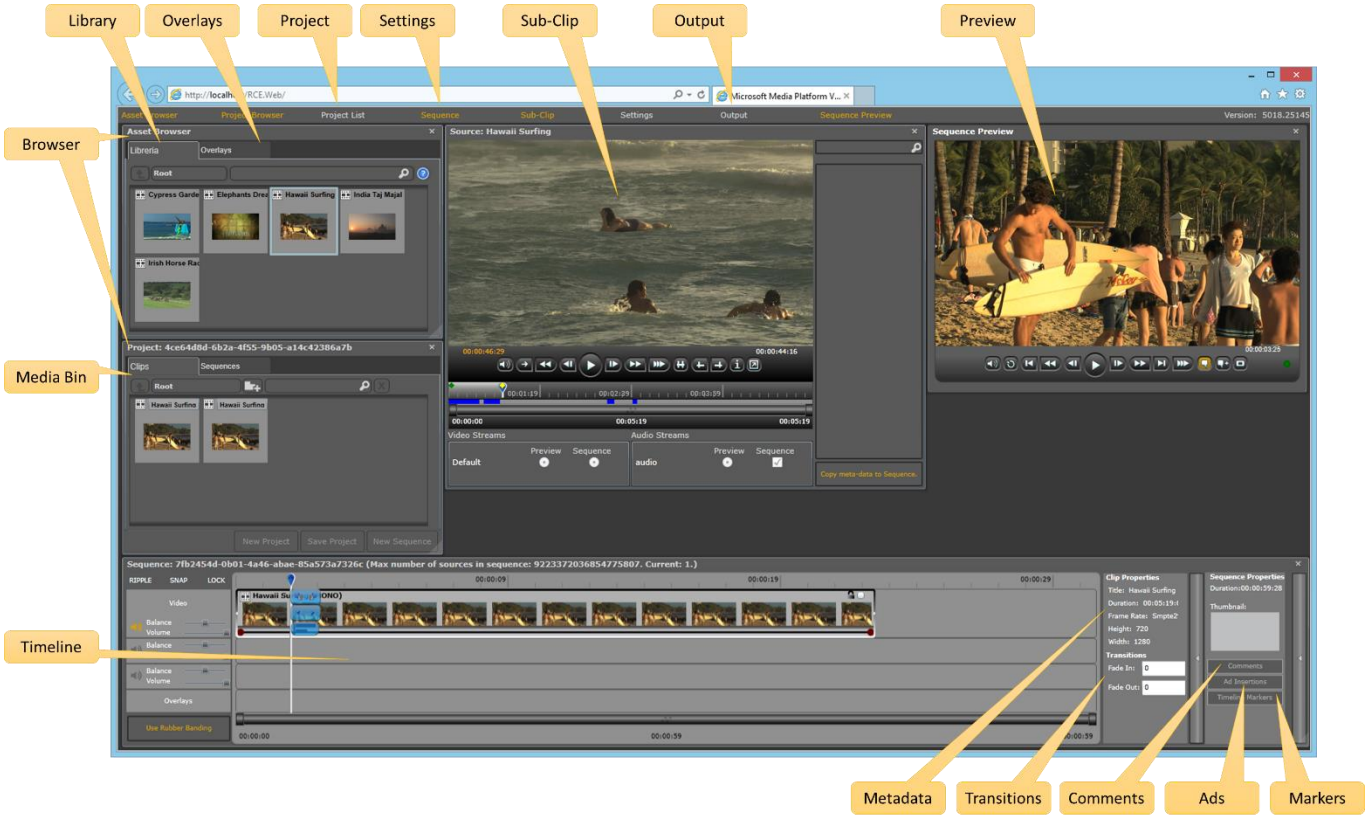
Logical Architecture Diagram

The following describes the main elements of the MMP Video Editor architecture:

- **Application**. The application is lightweight and contains the shell that hosts each of the different UI components within the application. It also contains the bootstrapper, which sets up the container and initializes module loading.
- **Modules**. The solution is divided into modules that can be maintained, tested and deployed independently.
- **Infrastructure**. The common infrastructure contains functionality for the MMP Video Editor and the Composite Application Guidance core.
- **Composite Application Library**. This contains the core composition services and service interfaces for handling regions, commanding, and module loading. It also contains the container façade for the Unity Application Block (also referred to as Unity).
- **RCE Infrastructure Library**. This contains service interfaces specific to the MMP Video Editor, shared models, and shared commands.

Modularity

A module is a logical unit of the application. Modules assist in implementing a modular design. These modules are defined in such a way that they can be discovered and loaded by the application at run time. Because modules are self-contained, they promote separation of concerns in the application. Modules can communicate with each other and access services in a loosely coupled fashion. They reduce the friction of maintaining, adding, and removing system functionality. Modules also aid in testing and deployment.



MMP Video Editor Modules Distribution

Playback Model

The Microsoft Media Platform Video Editor 2.0 playback model creates and reproduces a Composite Stream Manifest (CSM) per track, instead of using different Smooth Streaming Media Elements (SSMEs) to play each clip.

The reasons for this new model are:

- **Performance**: Previous version of the MMP Video Editor suffered from performance loss due to an increase in memory consumption when multiple clips were added to the timeline. The new playback model allows editors to place a very large number of clips in a sequence, without causing the memory consumption to grow, making the MMP Video Editor a responsive clip editing application.
- **Simplicity**: The new model requires simple logic to control the playback experience, in contrast with the old model which required many components to control how the timeline was reproduced.
- **WYSIWYG Experience**: Clips reproduced in the application's player that are created with the new playback model show editors the exact same content they will get when they export their sequences to a set of CSMs.

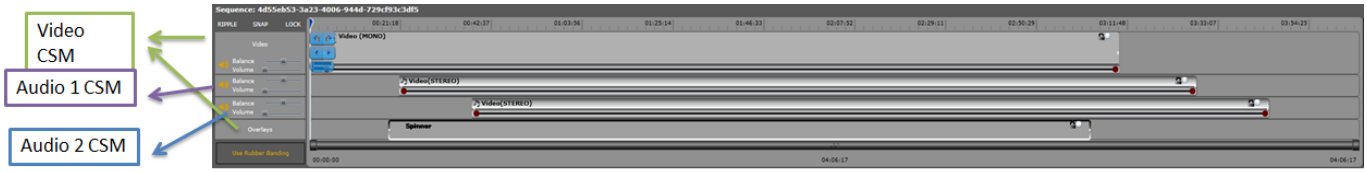
How does the Playback Model work?

The new playback model creates a Composite Stream Manifest per track. With the default configuration, which features Track #1 for both video and audio, and Tracks #2 and #3 for audio, three CSMs are created.

The content of each CSM is broken down below, accompanied by a figure:

- **Video CSM**. Includes video clips from Track #01. These video clips also have a single audio track, which is included as well. The Overlays track is also included.
- **Audio 1 CSM**. Includes the audio clips from Track #02
- **Audio 2 CSM**. Includes the audio clips from Track #03

Note: A CSM is only generated if its related track has clips.



Sequence Timeline

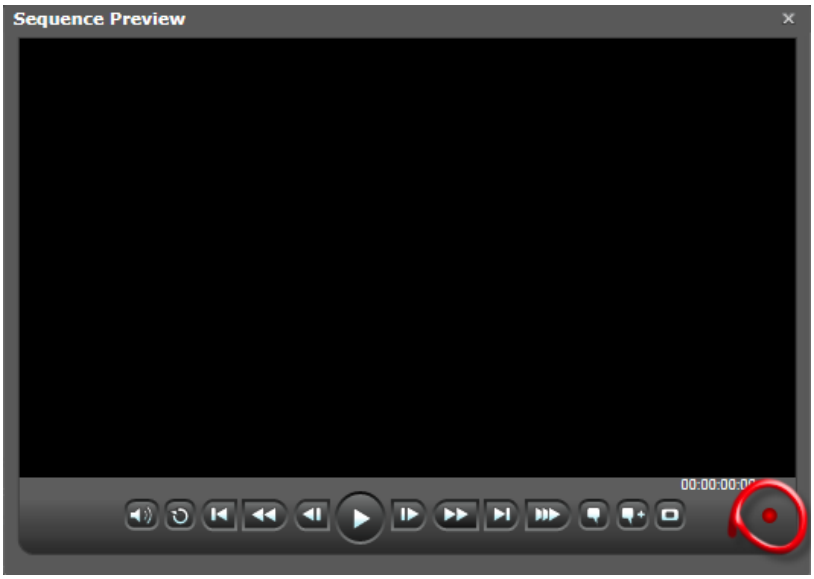
What triggers the CSMs generation?

Every time an editor performs a change that would require an update in a CSM, the timeline is considered to be "dirty." Operations that can cause the timeline to become dirty are:

- Adding/Deleting clips
- Moving clips
- Splitting clips
- Updating an overlay's properties
- Updating a clip's rubber-banding points

When the timeline becomes dirty, the light next to the **Sequence Preview** player turns red, as shown in the following figure.

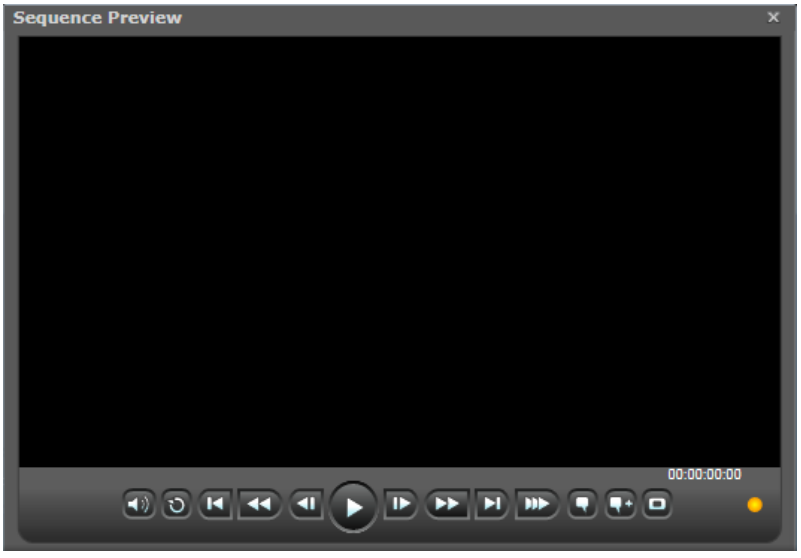
Note: If a sequence is being played and the timeline becomes dirty, playback is automatically paused.



Sequence Preview when Timeline is dirty

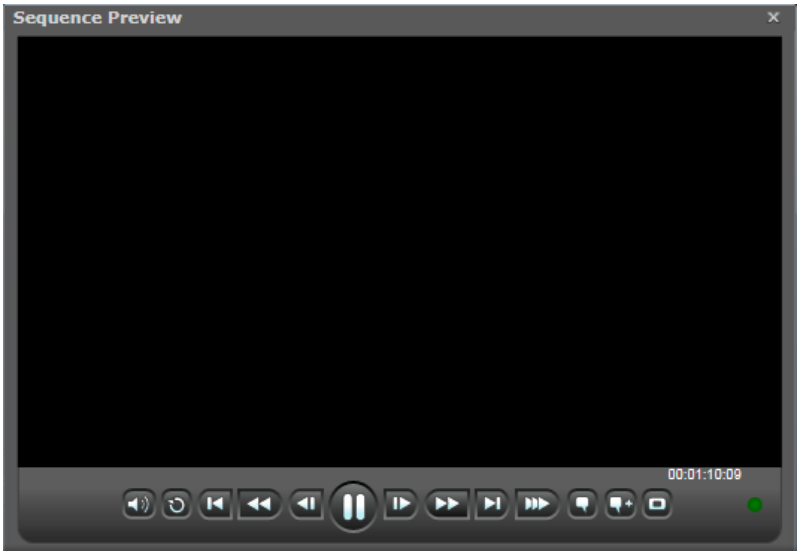
The red light is merely an indicator for the editor, which tells him/her that the sequence they are working on has some changes. To generate the new set of CSMs, and thus "clean" the sequence, you must click **Play**.

When you click **Play**, the indicator will turn yellow to notify users that the CSMs are being created. Additionally, **Play** will be disabled until the CSMs have been created. This can be seen in the following figure.



Sequence Preview when the CSMs are being created

Once the CSMs have been generated, the indicator will turn green to notify users that the sequence is "clean" and playback will begin. This is shown in the following figure.

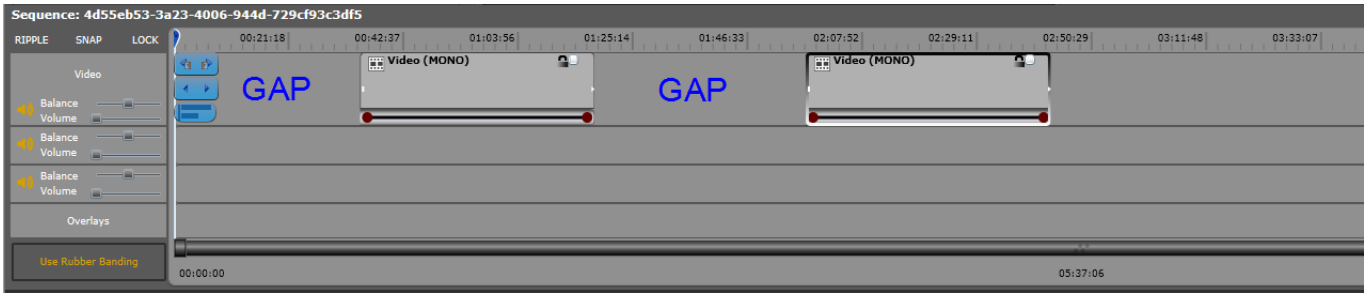


Sequence Preview ready

How are gaps between clips handled?

One of the interesting details about the new playback model is the way it handles gaps between clips. A gap is a sequence time interval that does not have any clips. The following figure shows the gaps in Track #1.

Note: The empty interval between the last clip and the end of the sequence is not considered a gap.



Sequence Timeline with gaps

The MMP Video Editor gives editors the flexibility of using their own video gaps, by configuring the video they want to use in the **Settings.xml** file.

XML

```
<Parameter Name="GapVideoUrl" Value="http://mydomain/media/gap.ism/manifest" />
```

The recommended practice is to use a muted video with a black overlay so it does not display any images or play any sounds while being played. The duration of this video is not important as the MMP Video Editor can include it as many times as necessary in the manifest to fill the duration of the gap.

Important Considerations

Due to a Silverlight limitation, audio encoding settings cannot change within a single CSM (you can find more information in this blog post). For that reason, make sure the gap video has the exact same audio encoding settings as the clips that are part of the timeline; otherwise, playback will not work.

The exception to this rule is having a sequence without gaps. In this case, if all clips within a single track share the same audio encoding settings, playback will work correctly. The URL for the gap video must be provided, but the gap will not be included in the resulting CSM.

Common Scenarios

This topic will describe common usage scenarios of the Microsoft Media Platform Video Editor with detailed steps of how to implement each of them.

SCENARIO A: I want to connect the MMP Video Editor to my CMS. The CMS already has services implemented and exposed

There are two options in order to connect the Microsoft Media Platform Video Editor to the existing services of your CMS depending on how your services were built.

If your services are Silverlight-enabled, you can create a new module on the Microsoft Media Platform Video Editor which will contain the implementation of the Microsoft Media Platform Video Editor services that can talk with the CMS services. In this module, you will register these new services to the container, overriding the default services registered by the MMP Video Editor. To populate the MMP Video Editor Media Library with the content retrieved by the CMS services, you will have to implement the **IAssetsDataServiceFacade**. To load and save projects on your CMS you will have to implement the **IDataServiceFacade**.

To implement this approach you will have to:

1. Create a new module on the MMP Video Editor and prepare it for remote downloading.
   Note: For generic instructions on how to create a module, see How to: Create a Module.
2. Register the new module in the module catalog used by the MMP Video Editor.
   Note: For generic instructions on how to register a new module in the module catalog, see How to: Populate the Module Catalog from XAML.
3. Implement your services and register them to the container.
   Note: For generic instructions on how to register services, see How to: Register and Use Services.

To create a module and register it in the module catalog:

1. Add a new Silverlight application to the MMP Video Editor solution to contain your module. When the Add Silverlight Application dialog box appears, do the following:
   a. Host the module on the same Web site as the main application. To do this, click **Link this Silverlight control into an existing Web site** and verify that the RCE.Web site is selected on the drop-down list.
   b. Clear the **Add a test page that references the application** check box, and then click OK.
2. Prepare the module for Remote Downloading. For more information on how to do this, see How to Prepare a module for Remote Downloading.
3. Add references to the following assemblies:
   a. Microsoft.Practices.Composite.dll
   b. **Microsoft.Practices.Composite.Unity.dll**
4. Add references to the following MMP Video Editor projects:
   a. RCE.Infrastructure
   b. RCE.Services.Contracts.Client
5. Delete the file **Class1.cs**
6. Add a new file named ***{ModuleName}*Module.cs** to your module project (replacing ***{ModuleName}*** with your module's name)
7. Add the following using statements at the top of file.

C#

```
using System;
using Microsoft.Practices.Composite.Modularity;
using Microsoft.Practices.Unity;
using RCE.Infrastructure;
using RCE.Infrastructure.Models;
```

8. Change the class signature to make it public and to implement the IModule interface, as shown in the following code.

C#

```
public class SampleModule : IModule
{
}
```

9. Implement the **Initialize** method of the **IModule** interface. In this method, you implement logic to initialize the module. In this scenario, you will register your services. The following code shows an example implementation of the Initialize method where an AssetsDataServiceFacade, a DataServiceFacade and a ThumbnailSevice are being registered to the container.

C#

```
public SampleModule(IUnityContainer container)
{
    this.container = container;
}

public void Initialize()
{
    this.container.RegisterType<IAssetsDataServiceFacade, MyAssetsDataServiceFacade>(new
ContainerControlledLifetimeManager());
    this.container.RegisterType<IDataServiceFacade, MyDataServiceFacade>(new
ContainerControlledLifetimeManager());
    this.container.RegisterType<IThumbnailService, MyThumbnailService>(new
ContainerControlledLifetimeManager());
}
```

Note: For more information about how to register services, see How to: Register and Use Services.

10. Configure the module so that it gets loaded when the application starts. To use modules in the MMP Video Editor, they have to be defined in the MMP Video Editor's module catalog. As the MMP Video Editor is using remote module loading, the modules have to be defined in the **ModulesCatalog.xaml** file located in the **RCE** project.
    Note: For more information about how to populate the module catalog from XAML, see How to: Populate the Module Catalog from XAML.

To implement an AssetsDataServiceFacade:

1. Add a new file to your module project.
2. Add the following using statements at the top of file.

C#

```
using System.Collections.Generic;
using RCE.Infrastructure;
using RCE.Infrastructure.Models;
```

3. Change the class signature to make it public and to implement the **IAssetsDataServiceFacade** interface, as shown in the following code.

```
C#
public class MyAssetsDataServiceFacade : IAssetsDataServiceFacade
{
}
```

4. Implement the methods of the **IAssetsDataServiceFacade** interface.

To implement a DataServiceFacade:

1. Add a new file to your module project.
2. Add the following using statements at the top of file.

```
C#
using System.Collections.Generic;
using RCE.Infrastructure;
using RCE.Infrastructure.Models;
```

3. Change the class signature to make it public and to implement the **IDataServiceFacade** interface, as shown in the following code.

```
C#
public class MyDataServiceFacade : IDataServiceFacade
{
}
```

4. Implement the methods of the **IDataServiceFacade** interface.

If your services are not Silverlight-enabled services, you will have to rely on the MMP Video Editor server side component to connect your CMS with the Microsoft Media Platform Video Editor. In this case you will have to implement the **IDataProvider** and the **IAssetsDataProvider** interfaces and consume your services on those implementations. Finally you will have to register both implementations on the **Web.config** to make them available for the application.

For more information about how to implement a Data Provider, see How to: Create a Data Provider.

For more information about how to implement an Assets Data Provider, see How to: Create an Assets Data Provider.

SCENARIO B: I want to connect the MMP Video Editor to my CMS. The CMS does not expose services.

In this scenario, as you don't have CMS services available, you can choose any of the approaches described in Scenario A. In both cases, you will have to implement the logic for CRUD operations against your CMS.

SCENARIO C: I want to load content to the MMP Video Editor library from a Web page in my CMS.

In this scenario, MMP Video Editor will get the media content for the library from a JavaScript bridge with a CMS Web page in XML format.

To implement this approach you will have to:

1. In your CMS page, invoke the **SearchServiceBridge** JavaScript bridge passing the XML with all the assets for your library.

HTML

```
<html>
<body>
 <script>
  function search() {
   var slControl = window.opener.document.getElementById("silverlightControl");
   var assetLibraryXml = "%the-xml-with-your-asset-library%";

   slControl.Content.SearchServiceBridge.SetSearchResults(assetLibraryXml);
   window.close();
  }
 </script>
 <div>
  <input type="button" value="Load Assets" onclick="javascript:search();" />
 </div>
</body>
</html>
```

2. The expected XML schema used in the **SearchServiceBridge** JavaScript bridge is the following.

XML

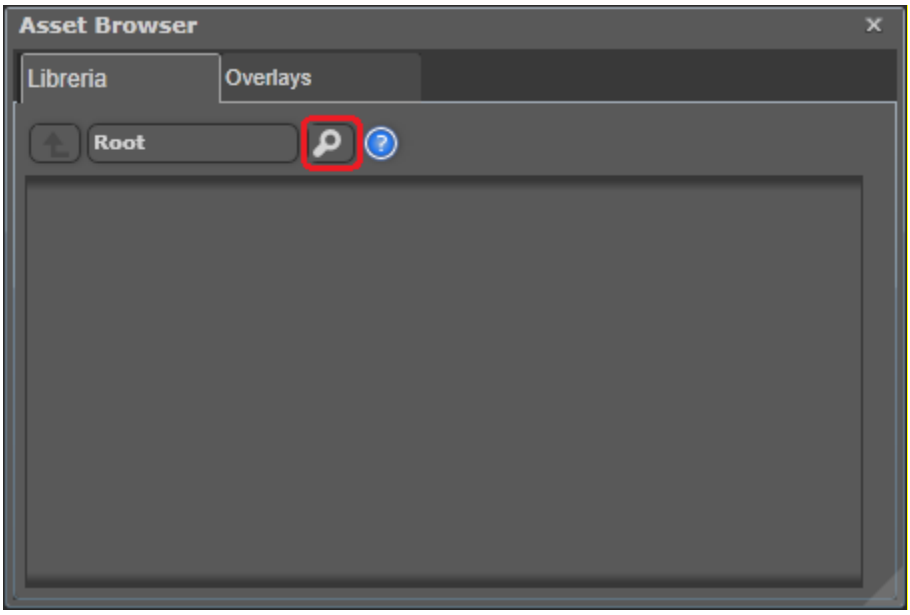
```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <count>2</count>
  <asset>
    <Title>Elephants Dream</Title>
    <Duration>654.446</Duration>

<Source>http://devplatem.vo.msecnd.net/RCEAzureDemo/ElephantsDream/ElephantsDream_VC1_EE4_VBR_10
80p_Xbox.ism/manifest</Source>

<wThumbnailPath>http://devplatem.vo.msecnd.net/RCEAzureDemo/thumbnails/elephantsdream.png</wThum
bnailPath>
    <KindName>Video</KindName>
  </asset>
  <asset>
    <Title>Fade Center</Title>
    <X>0.4</X>
    <Y>0.05</Y>
    <Height>0.2</Height>
    <Width>0.2</Width>
    <Duration>5</Duration>
    <KindName>Overlay</KindName>
    <Template>
      <!-- %Overlay's XAML code% -->
    </Template>
    <Fields>MainText,SubText</Fields>
  </asset>
</root>
```

3. In the **Settings.xml** file located in the **RCE.Web** project, change the value of the following settings accordingly.
   - **SearchIntegrationEnabled**. Set the value to **true**.
   - **SearchPopupUrl**. Replace with the URL of your CMS page which editors will use to filter and populate the assets' library.
   - **SearchPopupWidth**. Replace with the desired pop-up width for your CMS page.
   - **SearchPopupHeight**. Replace with the desired pop-up height for your CMS page.

XML

```
<Parameter Name="SearchIntegrationEnabled" Value="true" />
<Parameter Name="SearchPopupUrl" Value="http://localhost/RCE.Web/popup.html"/>
<Parameter Name="SearchPopupWidth" Value="800"/>
<Parameter Name="SearchPopupHeight" Value="200"/>
```

With this approach, you will need to click the **Search** button ( ) in MMP Video Editor to open your CMS Web page as a pop-up window. You can use the CMS page to filter the assets that you want to load in the library and then invoke the JavaScript bridge to send them back to MMP Video Editor in XML format.



Search button in MMP Video Editor

How to: Create a Data Provider

This topic describes how to create a data provider. Data Providers are used to perform different kind of operations such as loading and saving a project in the Microsoft Media Platform Video Editor.

Steps

Data Providers implement the **RCE.Services.Contracts.IDataProvider** interface. This interface defines the methods to load projects, save projects, and load a media bin among others.

The interface definition is show in the following code.

C#

```
public interface IDataProvider
{
    MediaBin LoadMediaBin(Uri mediaBinUri);

    TitleTemplateCollection LoadTitleTemplates();

    Project LoadProject(Uri site);

    bool SaveProject(Project project);

    ProjectCollection GetProjectsByUser(string userName);

    bool DeleteProject(Uri site);
}
```

The following procedure describes how to create a data provider.

To create a data provider:

1. Add a class library project to your solution to contain your data provider.
2. Add a reference to the **RCE.Services.Contracts.dll** assembly.
3. Add a new class to your project.
4. Add the following using statement at the top of the class file.

   C#

   ```
   using RCE.Services.Contracts;
   ```

5. Change your class signature to implement the **IDataProvider** interface as shown in the following code.

   C#

   ```
   public class MyCustomDataProvider : IDataProvider
   {
   }
   ```

6. The data providers are being created via dependency injection. One of the advantages that this approach provides is the ability to inject other registered dependencies into the class being created. Out of the box, the Microsoft Media Platform Video Editor provides a metadata locator and a logger as registered dependencies to inject. This is not restrictive as you can register many others dependencies or replace those that already exist.
   Note: For more information about the Dependency Injection pattern, check the More Information section below.
7. For example, to inject the logger, add the logger as a constructor parameter to the data provider as shown in the following code.

   C#

```
public MyCustomDataProvider(ILoggerService loggerService)
{
  // custom logic
}
```

8. Implement the **IDataProvider** methods.

To register the data provider:

1. Open the **Web.config** file.
2. Locate the **<unity>** section within the **Web.config** file. Under the **<containers>/<container>/<types>** subsection, the available RCE data providers are listed under the **<!-- Data Providers -->** comment.

XML

```
<unity>
    <typeAliases>
        ...
    </typeAliases>
    <containers>
        <container>
            <types>

            <!-- Data Providers -->

            <!-- Demo Data Provider -->
            <type type="RCE.Services.Contracts.IDataProvider, RCE.Services.Contracts"
                  mapTo="RCE.Data.Demo.DemoDataProvider, RCE.Data.Demo">
                <lifetime type="external"/>
            </type>
```

3. To register a new data provider, comment the existing data provider and add a type declaration under the **<unity>/<containers>/<container>/<types>** node as shown in the following code. Replace the highlighted sections with the created data provider information.

C#

```
<type type="RCE.Services.Contracts.IDataProvider, RCE.Services.Contracts"
      mapTo="CustomNamespace.CustomDataProviderClass, CustomAssembly">
    <lifetime type="external" />
</type>
```

4. Save and close the **Web.config** file.

Outcome

You will have a custom data provider created and registered. This data provider will be used by the RCE to perform operations such as save and load projects among others.

More Information

For more information about the Dependency Injection partner, see the following:

- Inversion of Control Containers and the Dependency Injection pattern on Martin Fowler's Web site
- Design Patterns: Dependency Injection by Griffin Caprio on MSDN

How to: Create an Asset Data Provider

This topic describes how to create an asset data provider. Asset Data Providers are used to retrieve the media items used on the Microsoft Media Platform Video Editor (MMP Video Editor).

Steps

Asset Data Providers implement the **RCE.Services.Contracts.IAssetsDataProvider** interface. This interface defines the methods to retrieve the media items being used on the Microsoft Media Platform Video Editor. The methods return a container instance containing the available media items. The interface definition is shown in the following code.

C#

```
public interface IAssetsDataProvider
{
    Container LoadLibrary(int maxNumberOfItems);

    Container LoadLibrary(string filter, int maxNumberOfItems);

    Container LoadLibraryById(Uri libraryId, int maxNumberOfItems)
}
```

The following procedure describes how to create an asset data provider.

To create an asset data provider:

1. Add a class library project to your solution. This project will contain your asset data provider.
2. Add a reference to the **RCE.Services.Contracts.dll** assembly.
3. Add a new class to your project.
4. Add the following **using** statement at the top of the class file.

   C#

   ```
   using RCE.Services.Contracts;
   ```

5. Change your class signature to implement the **IAssetsDataProvider** interface as shown in the following code.

   C#

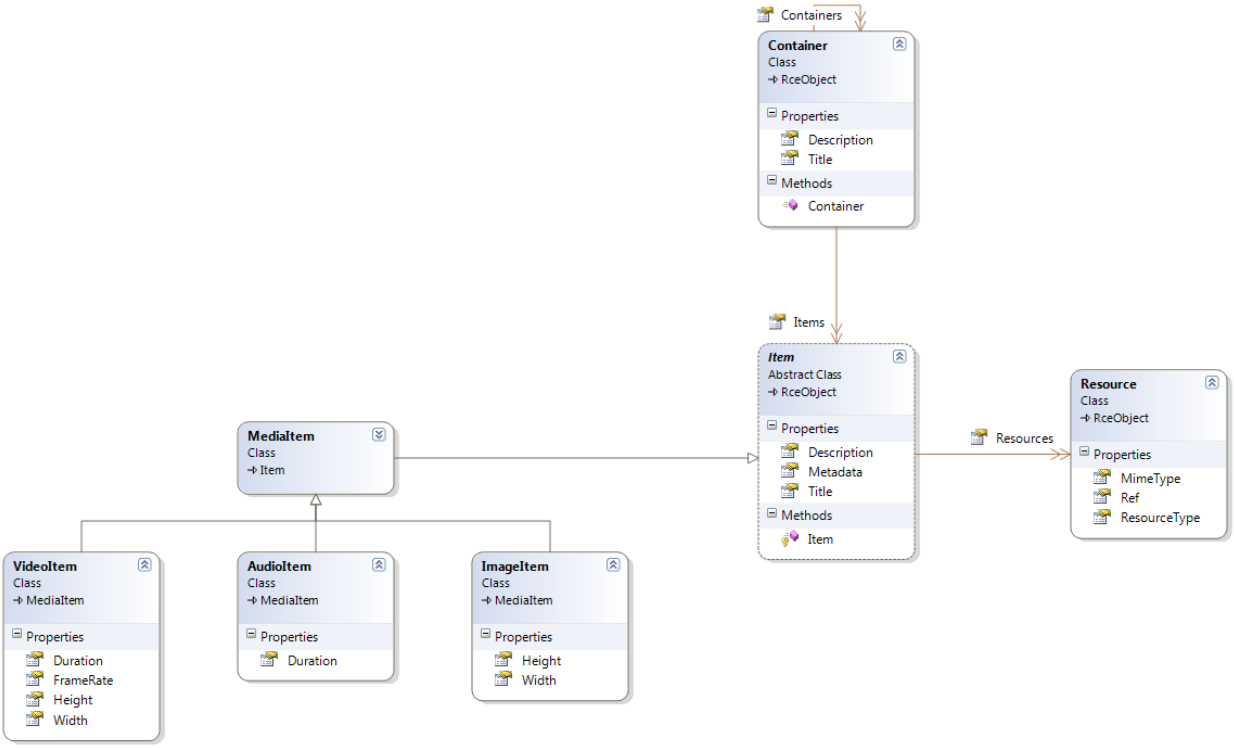
   ```
   public class MyCustomAssetsDataProvider : IAssetsDataProvider
   {
   }
   ```

6. The asset data providers are being created via dependency injection. One of the advantages that this approach provides is the ability to inject other registered dependencies into the class being created. Out of the box, the Microsoft Media Platform Video Editor provides a metadata locator and a logger as registered dependencies to inject. This is not restrictive as you can register many other dependencies or replace those that already exist.
   Note: For more information about the Dependency Injection pattern, check the More Information section below.
7. For example, to inject the metadata locator, add the metadata locator as a constructor parameter to the asset data provider as shown in the following code.

   C#

   ```
   public MyCustomAssetsDataProvider(IMetadataLocator metadataLocator)
   {
     // custom logic
   }
   ```

8. Implement the **LoadLibrary** methods. The methods should return a container instance containing the media items retrieved. The methods take different parameters depending on the overload:
    - **containerUri**. The Uri of the base container containing items
    - **maxNumberOfItems**. The max number of items to be retrieved. This value is set in the Microsoft Media Platform Video Editor configuration.
    - **filter**. A filter term used to search for items
9. The following diagram shows the class model of a container and its items.



Container Model

To register the asset data provider:

1. Open the Web.config file.
2. Locate the **<unity>** section within the **Web.config** file. Under the **<containers>/<container>/<types>** subsection, the available MMP Video Editor asset data providers are listed under the **<!—Assets Data Providers -->** comment.

XML

```
<!-- Assets Data Providers -->

<!-- Demo Assets Data Provider -->
<type type="RCE.Services.Contracts.IAssetsDataProvider, RCE.Services.Contracts"
      mapTo="RCE.Data.Demo.DemoDataProvider, RCE.Data.Demo">
  <lifetime type="external"/>
</type>
```

3. To register a new asset data provider, comment the existing asset data provider and add a type declaration under the **<unity>/<containers>/<container>/<types>** node as shown in the following code. Replace the highlighted sections with the created asset data provider information.

XML

```
<type type="RCE.Services.Contracts.IAssetsDataProvider, RCE.Services.Contracts"
      mapTo="CustomNamespace.CustomAssetsDataProviderClass, CustomAssembly">
    <lifetime type="external" />
</type>
```

4. Save and close the **Web.config** file.

Outcome

You will have a custom asset data provider created and registered. This asset data provider will be used to populate the MMP Video Editor with the media items.

More Information

For more information about the Dependency Injection partner, see the following:

- Inversion of Control Containers and the Dependency Injection pattern on Martin Fowler's Web site
- Design Patterns: Dependency Injection by Griffin Caprio on MSDN

How to: Create a Metadata Strategy

This topic describes how to create a metadata strategy. Metadata strategies are used to retrieve metadata associated to media items.

Steps

Metadata strategies implement the **RCE.Services.Contracts.IMetadataStrategy** interface. This interface defines two methods. One is named **CanRetrieveMetadata** that takes the object from which to extract the metadata and returns a Boolean value indicating whether or not the strategy can retrieve the metadata from the object. The other is named **GetMetadata** and takes the object from which to extract the metadata and returns the metadata associated with the object. The following code illustrates the **IMetadataStrategy** definition.

C#

```
public interface IMetadataStrategy
{
    bool CanRetrieveMetadata(object target);

    Metadata GetMetadata(object target);
}
```

The following procedure describes how to create a metadata strategy.

To create a metadata strategy:

1. Add a class library project to your solution. This project will contain your metadata strategy.
2. Add a reference to the **RCE.Services.Contracts.dll** assembly.
3. Add a new class to your project.
4. Add the following using statement at the top of the class file.

   C#

   ```
   using RCE.Services.Contracts;
   ```

5. Change your class signature to implement the **IMetadataStrategy** interface as shown in the following code.

   C#

   ```
   public class MyCustomMetadataStrategy : IMetadataStrategy
   {
   }
   ```

6. Implement the **CanRetrieveMetadata** method. This method returns a Boolean value that indicates if the metadata strategy can retrieve the metadata of the target object. In general, the target object is a media item object, but it can be another kind of object (such us a string indicating the full path of the media item).

   C#

   ```
   public bool CanRetrieveMetadata(object target)
   {
       // replace with your own logic.
       return true;
   }
   ```

7. Implement the **GetMetadata** method. This method should return a metadata instance (an object that inherits from the **RCE.Services.Contracts.Metadata** base class).

   C#

   ```
   public Metadata GetMetadata(object target)
   ```

```
{
    MyCustomMetadata metadata = new MyCustomMetadata();
    // replace with your own logic.
    return metadata;
}
```

Metadata strategies are used by the metadata locator service to locate the correct metadata strategy for a target object. The following procedure describes how to register the metadata strategy created in the aforementioned procedure.

To register the metadata strategy:

1. Open the **Web.config** file.
2. Locate the **<unity>** section within the **Web.config** file. Under the **<containers>/<container>/<types>** subsection, the available MMP Video Editor metadata strategies are listed.
   Note: The listed strategies can be removed without any risk.

```
<!-- Metadata Strategies -->

  <type type="RCE.Services.Contracts.IMetadataStrategy, RCE.Services.Contracts"
        mapTo="RCE.Metadata.Strategies.WMMetadataStrategy, RCE.Metadata.Strategies" name="WMMetadataStrategy">
    <lifetime type="external" />
  </type>

  <type type="RCE.Services.Contracts.IMetadataStrategy, RCE.Services.Contracts"
        mapTo="RCE.Metadata.Strategies.ImageMetadataStrategy, RCE.Metadata.Strategies" name="ImageMetadataStrategy">
    <lifetime type="external" />
  </type>
```

3. To register a new strategy, add a type declaration under the **<unity>/<containers>/<container>/<types>** node as shown in the following code. Replace the highlighted sections with the created metadata strategy information.

XML

```
<type type="RCE.Services.Contracts.IMetadataStrategy, RCE.Services.Contracts"
      mapTo="CustomNamespace.CustomMetadataStrategyClass, CustomAssembly"
      name="CustomMetadataName">
    <lifetime type="external" />
</type>
```

4. Save and close the **Web.config** file.

Outcome

You will have a custom metadata strategy created and registered. This metadata strategy will be used by the metadata locator to retrieve media items metadata.

How to: Create a Timeline Bar Element

This topic describes how to create a Timeline Bar Element. Timeline Bar elements are used to highlight particular events within the timeline. The MMP Video Editor provides these types of Timeline Bar elements out of the box:

- Comments
- Ad Insertions
- Markers
    - Play by Play

Steps

Timeline Bar elements are usually parts of a module registered in the TimelineBarRegistry service. The TimelineBarRegistry service is the component that keeps track of all the available kinds of Timeline Bar Elements.

A Timeline Bar element is normally divided into 4 components:

- **The Edit Box view**. This is the main view where the major operations of the timeline bar element can be performed, such as save, delete and edit.
- **The Preview view**. This is the preview that will appear on the timeline once the timeline bar element is added.
- **The Display Box view**. This is the read-only view that displays the information of the timeline bar element.
- **The element or model**. This is what the timeline bar element is representing.

The views are associated with a ViewModel which will manage the interaction between them and the timeline bar. Timeline Bar elements ViewModels implement the **RCE.Infrastructure.Models.ITimelineBarElementModel** interface. This interface defines two events, four properties and four methods:

Events

- **TimelineBarElementUpdated**. This event is intended to be raised after an element has been updated. The Timeline Bar will reflect the changes after handling this event.
- **Deleting**. This event is intended to be raised after an element has been deleted. The Timeline Bar will reflect the changes after handling this event.

Properties

- **EditBox**. This property is intended to return the view that is going to be used as the EditBox. The property is of object type to allow flexibility in how this view is created.
- **Preview**. This property is intended to return the view that is going to be used as the Preview. The property is of object type to allow flexibility in how this view is created.
- **Position**. This property is intended to return the current position in seconds of the element/model.
- **DisplayBox**. This property is intended to return the view that is going to be used as DisplayBox. The property is of object type to allow flexibility in how this view is created.

Methods

- **SetPosition**. This method is called when an external component wants to set a new position to the timeline bar element. This commonly happens when the timeline bar element is added or updated. This method is intended to set the new position to the timeline bar element and the model.
- **RefreshPreview**. This method is called when changes occur on the timeline. This method is intended to reposition the preview.
- **SetElement**. This method is called when an external component wants to set or replace the element being used by the ITimelineBarElement.
- **GetElement**. This method is called when an external component wants to retrieve the element being used by the ITimelineBarElement.

The interface definition is shown in the following code.

C#

```
public interface ITimelineBarElementModel
{
    event EventHandler<EventArgs> TimelineBarElementUpdated;

    event EventHandler<EventArgs> Deleting;

    object EditBox { get; }

    object Preview { get; }

    double Position { get; }

    object DisplayBox { get; }

    void SetPosition(TimeSpan position);

    void RefreshPreview(double refreshedWidth);

    void SetElement(object value, CommentMode mode);

    T GetElement<T>() where T: class;
}
```

The following procedure describes how to create a timeline bar element.

To create a timeline bar element:

1. Add a new class to the module project to which the timeline bar element will belong.
2. Add the following **using** statement at the top of the class file.

   C#

   ```
   using RCE.Infrastructure.Models;
   ```

3. Change the signature of the class to implement the **ITimelineBarElementModel** interface, as shown in the following code.

   C#

   ```
   public class MyCustomTimelineBarElementPresentationModel : ITimelineBarElementModel
   {
   }
   ```

4. Implement the interface following the guidelines explained in the Steps section.
5. Save the class.

6. Create the EditBox view, the Preview view and the DisplayBox view.
   Note: In order to understand how these views should be injected into the Timeline Bar Element Presentation Model, please see the examples provided in the MMP Video Editor (Comments, Ads, Markers or PlayByPlay).

Timeline Bar elements are registered with the TimelineBarRegistry service. The following procedure describes how to register a timeline bar element.

To register a timeline bar element:

1. Open the module class file to which the Timeline Bar Element belongs.
2. In the **Initialize** method, retrieve the TimelineBarRegistry from the container and register the timeline bar element to it, as shown in the highlighted code:

C#

```
public void Initialize()
{
    this.RegisterViewsAndServices();

    ITimelineBarRegistry registry = this.container.Resolve<ITimelineBarRegistry>();

    registry.RegisterTimelineBarElement("NameOfTheTimelineBarElement", () =>
this.container.Resolve<MyCustomTimelineBarElementPresentationModel>());
}
```

3. Save and close the module class file.

Outcome

You will have a custom timeline bar element crated and registered. This timeline bar element will be available to be added on the timeline bar.