

Contents

Media Services Documentation

Overview

- About Media Services

Quickstarts

- Stream video files with .NET

Tutorials

- Encode remote file and stream video – REST

- Encode uploaded file and stream video - .NET

- Stream live - .NET

- Analyze your video - .NET

Samples

- Azure code samples

- Azure CLI

Concepts

- Migrate from v2 to v3

- Analyzing content

- Encoding content

 - Media Encoder standard formats

 - Autogen adaptive bitrate ladder

- Examining entities

 - Account and Asset filters

 - Assets

 - Content key policies

 - Streaming endpoints

 - Streaming locators

 - Streaming policies

 - Transforms and Jobs

- Examining storage accounts

- Handling Event Grid events

Protecting content

Overview

Multi-DRM system with access control

Streaming live events

Recommended live encoders

Using a cloud DVR

Feature comparison

States and billing

Latency

How-to guides

Create an account - CLI

Access APIs - CLI

Upload

From HTTP(s) - .NET

From a local file - .NET

From a local file - REST

Create filters

With .NET

With REST

With CLI

Encode

Customize Transform presets - .NET Core

Use Event Grid

Monitor events - Portal

Monitor events - CLI

Protect

With AES-128 - .NET

With DRM - .NET

Configure Postman

Scale Media Reserved Units - CLI

Reference

OpenAPI Specification (Swagger)

SDKs/Tools

[REST](#)

[Azure CLI](#)

[.NET](#)

[Java](#)

[Python](#)

[Node.js](#)

[Go](#)

Event Grid schemas

DRM licenses

[PlayReady license template](#)

[Widevine license template](#)

[FairPlay license requirements](#)

Resources

[Azure Roadmap](#)

[Pricing](#)

[Regional availability](#)

[Videos](#)

[OpenAPI Specification \(Swagger\)](#)

SDKs/Tools

[Azure CLI](#)

[.NET](#)

[Java](#)

[Python](#)

[Node.js](#)

[Ruby](#)

[Release notes](#)

[Quotas and limitations](#)

[FAQ](#)

What is Azure Media Services v3?

12/14/2018 • 5 minutes to read • [Edit Online](#)

Azure Media Services is a cloud-based platform that enables you to build solutions that achieve broadcast-quality video streaming, enhance accessibility and distribution, analyze content, and much more. Whether you are an application developer, a call center, a government agency, an entertainment company, Media Services helps you create applications that deliver media experiences of outstanding quality to large audiences on today's most popular mobile devices and browsers.

What can I do with Media Services?

Media Services enables you to build a variety of media workflows in the cloud, the following are some examples of what can be accomplished with Media Services.

- Deliver videos in various formats so they can be played on a wide variety of browsers and devices. For both on-demand and live streaming delivery to various clients (mobile devices, TV, PC, etc.) the video and audio content needs to be encoded and packaged appropriately. To see how to deliver and stream such content, see [Quickstart: Encode and stream files](#).
- Stream live sporting events to a large online audience, such as soccer, baseball, college and high school sports, and more.
- Broadcast public meetings and events such as town halls, city council meetings, and legislative bodies.
- Analyze recorded videos or audio content. For example, to achieve higher customer satisfaction, organizations can extract speech-to-text and build search indexes and dashboards. Then, they can extract intelligence around common complaints, sources of complaints, and other relevant data.
- Create a subscription video service and stream DRM protected content when a customer (for example, a movie studio) needs to restrict the access and use of proprietary copyrighted work.
- Deliver offline content for playback on airplanes, trains, and automobiles. A customer might need to download content onto their phone or tablet for playback when they anticipate to be disconnected from the network.
- Add subtitles and captions to videos to cater to a broader audience (for example, people with hearing disabilities or people who want to read along in a different language).
- Implement an educational e-learning video platform with Azure Media Services and [Azure Cognitive Services APIs](#) for speech-to-text captioning, translating to multi-languages, etc.
- Enable Azure CDN to achieve large scaling to better handle instantaneous high loads (for example, the start of a product launch event.)

v3 capabilities

v3 is based on a unified API surface, which exposes both management and operations functionality built on Azure Resource Manager.

This version provides the following capabilities:

- **Transforms** that help you define simple workflows of media processing or analytics tasks. Transform is a recipe for processing your video and audio files. You can then apply it repeatedly to process all the files in your content library, by submitting jobs to the Transform.
- **Jobs** to process (encode or analyze) your videos. An input content can be specified on a job using HTTPS URLs, SAS URLs, or paths to files located in Azure Blob storage. Currently, AMS v3 does not support chunked transfer encoding over HTTPS URLs.
- **Notifications** that monitor job progress or states, or Live Channel start/stop and error events. Notifications

are integrated with the Azure Event Grid notification system. You can easily subscribe to events on several resources in Azure Media Services.

- **Azure Resource Management** templates can be used to create and deploy Transforms, Streaming Endpoints, Channels, and more.
- **Role-based access control** can be set at the resource level, allowing you to lock down access to specific resources like Transforms, Channels, and more.
- **Client SDKs** in multiple languages: .NET, .NET core, Python, Go, Java, and Nodejs.

Naming conventions

Azure Media Services v3 resource names (for example, Assets, Jobs, Transforms) are subject to Azure Resource Manager naming constraints. In accordance with Azure Resource Manager, the resource names are always unique. Thus, you can use any unique identifier strings (for example, GUIDs) for your resource names.

Media Services resource names cannot include: '<', '>', '%', '&', ':', '\', '?', '/', '*', '+', ',', the single quote character, or any control characters. All other characters are allowed. The max length of a resource name is 260 characters.

For more information about Azure Resource Manager naming, see: [Naming requirements](#) and [Naming conventions](#).

Media Services v3 API design principles

One of the key design principles of the v3 API is to make the API more secure. v3 APIs do not return secrets or credentials on a **Get** or **List** operation. The keys are always null, empty, or sanitized from the response. You need to call a separate action method to get secrets or credentials. Separate actions enable you to set different RBAC security permissions in case some APIs do retrieve/display secrets while other APIs do not. For information on how to manager access using RBAC, see [Use RBAC to manage access](#).

Examples of this include

- not returning ContentKey values in the Get of the StreamingLocator,
- not returning the restriction keys in the get of the ContentKeyPolicy,
- not returning the query string part of the URL (to remove the signature) of Jobs' HTTP Input URLs.

The following .NET example shows how to get a signing key from the existing policy. You need to use **GetPolicyPropertiesWithSecretsAsync** to get to the key.

```

private static async Task<ContentKeyPolicy> GetOrCreateContentKeyPolicyAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string contentKeyPolicyName)
{
    ContentKeyPolicy policy = await client.ContentKeyPolicies.GetAsync(resourceGroupName, accountName,
contentKeyPolicyName);

    if (policy == null)
    {
        // Configure and create a new policy.

        . . .
        policy = await client.ContentKeyPolicies.CreateOrUpdateAsync(resourceGroupName, accountName,
contentKeyPolicyName, options);
    }
    else
    {
        var policyProperties = await
client.ContentKeyPolicies.GetPolicyPropertiesWithSecretsAsync(resourceGroupName, accountName,
contentKeyPolicyName);
        var restriction = policyProperties.Options[0].Restriction as ContentKeyPolicyTokenRestriction;
        if (restriction != null)
        {
            var signingKey = restriction.PrimaryVerificationKey as ContentKeyPolicySymmetricTokenKey;
            if (signingKey != null)
            {
                TokenSigningKey = signingKey.KeyValue;
            }
        }
    }

    return policy;
}

```

How can I get started with v3?

As a developer, you can use Media Services [REST API](#) or client libraries that allow you to interact with the REST API, to easily create, manage, and maintain custom media workflows.

Media Services provides [Swagger files](#) that you can use to generate SDKs for your preferred language/technology.

Microsoft generates and supports the following client libraries:

API REFERENCES	SDKS/TOOLS	EXAMPLES
REST ref	REST SDK	REST Postman examples Azure Resource Manager based REST API
Azure CLI ref	Azure CLI	Azure CLI examples
.NET ref	.NET SDK	.NET examples
	.NET Core SDK (Choose the .NET CLI tab)	.NET Core examples
Java ref	Java SDK	

API REFERENCES	SDKS/TOOLS	EXAMPLES
Node.js ref	Node.js SDK	Node.js samples
Python ref	Python SDK	
Go ref	Go SDK	
Ruby	Ruby SDK	

Next steps

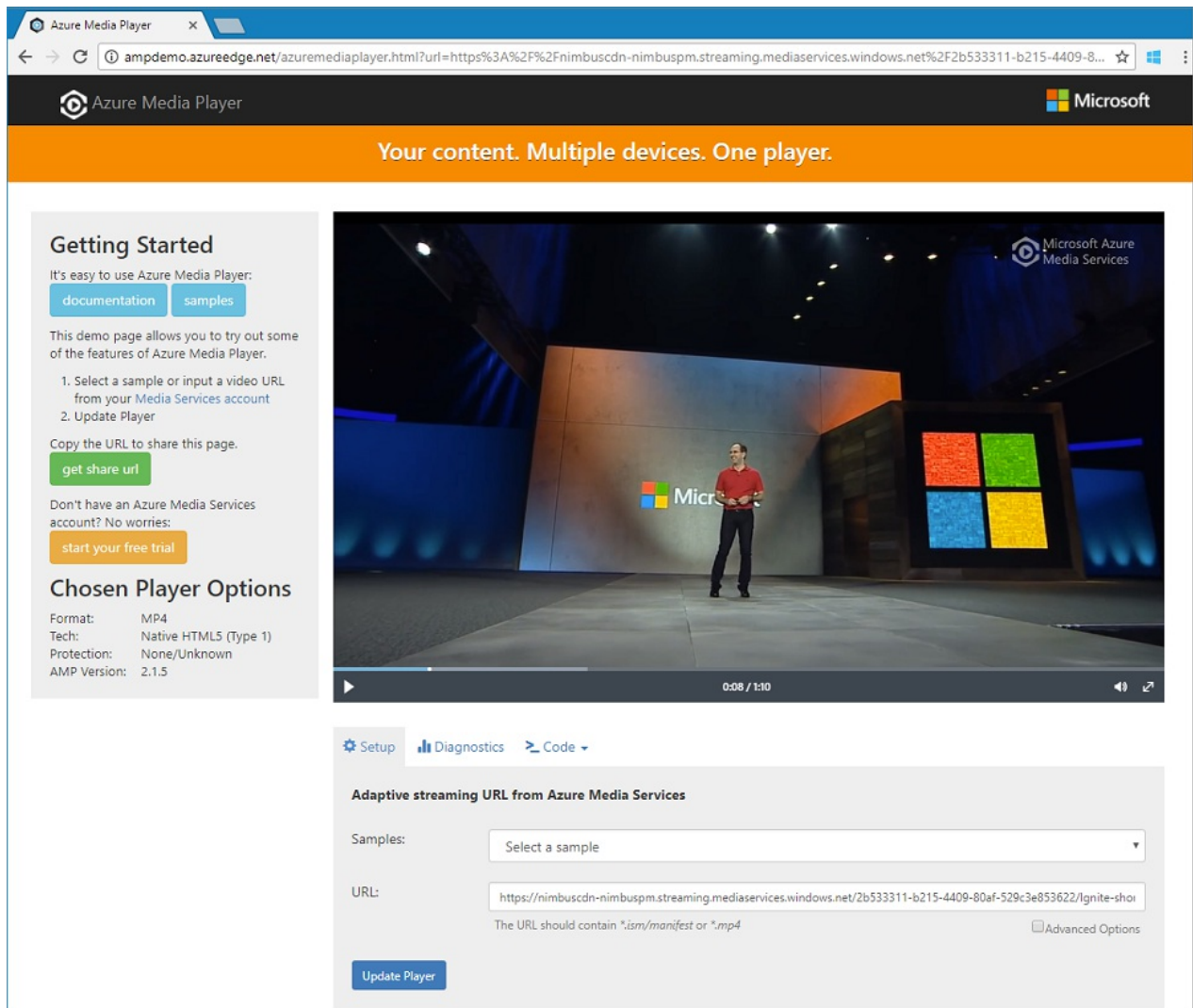
To see how easy it is to start encoding and streaming video files, check out [Stream files](#).

Quickstart: Stream video files - .NET

11/13/2018 • 3 minutes to read • [Edit Online](#)

This quickstart shows you how easy it is to encode and start streaming videos on a wide variety of browsers and devices using Azure Media Services. An input content can be specified using HTTPS URLs, SAS URLs, or paths to files located in Azure Blob storage. The sample in this topic encodes content that you make accessible via an HTTPS URL. Note that currently, AMS v3 does not support chunked transfer encoding over HTTPS URLs.

By the end of the quickstart you will be able to stream a video.



If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- If you do not have Visual Studio installed, you can get [Visual Studio Community 2017](#).
- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#).

Download the sample

Clone a GitHub repository that contains the streaming .NET sample to your machine using the following command:

```
git clone http://github.com/Azure-Samples/media-services-v3-dotnet-quickstarts.git
```

The sample is located in the [EncodeAndStreamFiles](#) folder.

The sample performs the following actions:

1. Creates a Transform (first, checks if the specified Transform exists).
2. Creates an output Asset that is used as the encoding Job's output.
3. Creates the Job's input that is based on an HTTPS URL.
4. Submits the encoding Job using the input and output that was created earlier.
5. Checks the Job's status.
6. Creates a StreamingLocator.
7. Builds streaming URLs.

For explanations about what each function in the sample does, examine the code and look at the comments in [this source file](#).

Access the Media Services API

To connect to Azure Media Services APIs, you use the Azure AD service principal authentication. The following command creates an Azure AD application and attaches a service principal to the account. You should use the returned values to configure your application.

Before running the script, you can replace the `amsaccount` and `amsResourceGroup` with the names you chose when creating these resources. `amsaccount` is the name of the Azure Media Services account where to attach the service principal.

The following command returns a `json` output:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup
```

This command produces a response similar to this:

```
{
  "AadClientId": "00000000-0000-0000-0000-000000000000",
  "AadEndpoint": "https://login.microsoftonline.com",
  "AadSecret": "00000000-0000-0000-0000-000000000000",
  "AadTenantId": "00000000-0000-0000-0000-000000000000",
  "AccountName": "amsaccount",
  "ArmAadAudience": "https://management.core.windows.net/",
  "ArmEndpoint": "https://management.azure.com/",
  "Region": "West US 2",
  "ResourceGroup": "amsResourceGroup",
  "SubscriptionId": "00000000-0000-0000-0000-000000000000"
}
```

If you would like to get an `xml` in the response, use the following command:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup --xml
```

Run the sample app

When you run the app, URLs that can be used to playback the video using different protocols are displayed.

1. Press Ctrl+F5 to run the *EncodeAndStreamFiles* application.
2. Choose the Apple's **HLS** protocol (ends with *manifest(format=m3u8-aapl)*) and copy the streaming URL from the console.

```
CA\WINDOWS\system32\cmd.exe
Job is Queued.
  JobOutput[0] is Queued.
Job is Processing.
  JobOutput[0] is Processing. Progress: 0
Job is Processing.
  JobOutput[0] is Processing. Progress: 24
Job is Processing.
  JobOutput[0] is Processing. Progress: 49
Job is Processing.
  JobOutput[0] is Processing. Progress: 76
Downloading results to D:\AMS\output\d894fd92-a050-45e7-871a-34a230745d5c-output.
Download complete.
http://amstest.streaming.mediaservices.windows.net/d4acbbdf-657a-4954-b7ce-d7c85c8b4796/ignite-short.ism/manifest(format=m3u8-aapl)
http://amstest.streaming.mediaservices.windows.net/d4acbbdf-657a-4954-b7ce-d7c85c8b4796/ignite-short.ism/manifest(format=mpd-time-csf)
http://amstest.streaming.mediaservices.windows.net/d4acbbdf-657a-4954-b7ce-d7c85c8b4796/ignite-short.ism/manifest
Press any key to continue . . .
```

In the sample's [source code](#), you can see how the URL is built. To build it, you need to concatenate the streaming endpoint's host name and the streaming locator path.

Test with Azure Media Player

To test the stream, this article uses Azure Media Player.

NOTE

If a player is hosted on an https site, make sure to update the URL to "https".

1. Open a web browser and navigate to <https://aka.ms/azuremediaplayer/>.
2. In the **URL:** box, paste one of the streaming URL values you got when you ran the application.
3. Press **Update Player**.

Azure Media Player can be used for testing but should not be used in a production environment.

Clean up resources

If you no longer need any of the resources in your resource group, including the Media Services and storage accounts you created for this Quickstart, delete the resource group.

Execute the following CLI command:

```
az group delete --name amsResourceGroup
```

Examine the code

For explanations about what each function in the sample does, examine the code and look at the comments in [this source file](#).

The [upload, encode, and stream files](#) tutorial gives you a more advanced streaming example with detailed explanations.

Multithreading

The Azure Media Services v3 SDKs are not thread-safe. When working with multi-threaded application, you should generate a new AzureMediaServicesClient object per thread.

Next steps

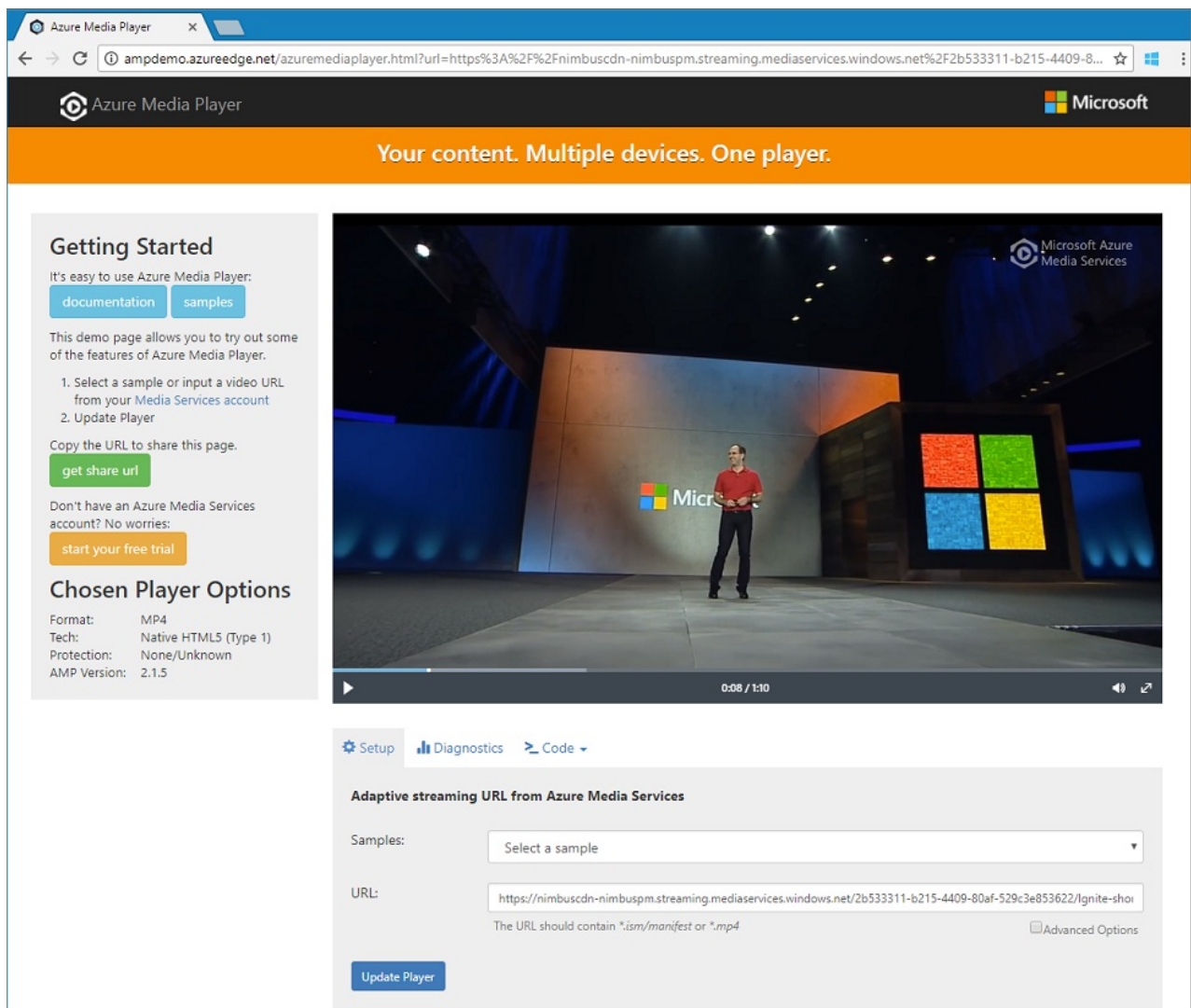
[Tutorial: upload, encode, and stream files](#)

Tutorial: Encode a remote file based on URL and stream the video - REST

12/18/2018 • 9 minutes to read • [Edit Online](#)

Azure Media Services enables you to encode your media files into formats that can be played on a wide variety of browsers and devices. For example, you might want to stream your content in Apple's HLS or MPEG DASH formats. Before streaming, you should encode your high-quality digital media file. For encoding guidance, see [Encoding concept](#).

This tutorial shows you how to encode a file based on a URL and stream the video with Azure Media Services using REST.



The screenshot displays the Azure Media Player interface in a web browser. The browser's address bar shows the URL: `ampdemo.azureedge.net/azuremediaplayer.html?url=https%3A%2F%2Fnimbuscdn-nimbuspm.streaming.mediaservices.windows.net%2F2b533311-b215-4409-80af-529c3e853622/ignite-shoi`. The player's header includes the Azure Media Player logo and the Microsoft logo, with the tagline "Your content. Multiple devices. One player." Below this, the "Getting Started" section provides instructions on how to use the player, including links to documentation and samples. A video player is shown with a video of a person on a stage. Below the video player, the "Chosen Player Options" section lists the format (MP4), tech (Native HTML5 (Type 1)), protection (None/Unknown), and AMP version (2.1.5). The "Adaptive streaming URL from Azure Media Services" section includes a "Samples" dropdown menu, a "URL" input field containing the same URL as the browser's address bar, and an "Update Player" button. The URL input field has a note: "The URL should contain *.ism/manifest or *.mp4".

This tutorial shows you how to:

- Create a Media Services account
- Access the Media Services API
- Download Postman files
- Configure Postman
- Send requests using Postman
- Test the streaming URL

- Clean up resources

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#).

Make sure to remember the values that you used for the resource group name and Media Services account name

- Install the [Postman](#) REST client to execute the REST APIs shown in some of the AMS REST tutorials.

We are using **Postman** but any REST tool would be suitable. Other alternatives are: **Visual Studio Code** with the REST plugin or **Telerik Fiddler**.

Download Postman files

Clone a GitHub repository that contains the Postman collection and environment files.

```
git clone https://github.com/Azure-Samples/media-services-v3-rest-postman.git
```

Access the Media Services API

To connect to Azure Media Services APIs, you use the Azure AD service principal authentication. The following command creates an Azure AD application and attaches a service principal to the account. You should use the returned values to configure your application.

Before running the script, you can replace the `amsaccount` and `amsResourceGroup` with the names you chose when creating these resources. `amsaccount` is the name of the Azure Media Services account where to attach the service principal.

The following command returns a `json` output:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup
```

This command produces a response similar to this:

```
{
  "AadClientId": "00000000-0000-0000-0000-000000000000",
  "AadEndpoint": "https://login.microsoftonline.com",
  "AadSecret": "00000000-0000-0000-0000-000000000000",
  "AadTenantId": "00000000-0000-0000-0000-000000000000",
  "AccountName": "amsaccount",
  "ArmAadAudience": "https://management.core.windows.net/",
  "ArmEndpoint": "https://management.azure.com/",
  "Region": "West US 2",
  "ResourceGroup": "amsResourceGroup",
  "SubscriptionId": "00000000-0000-0000-0000-000000000000"
}
```

If you would like to get an `xml` in the response, use the following command:

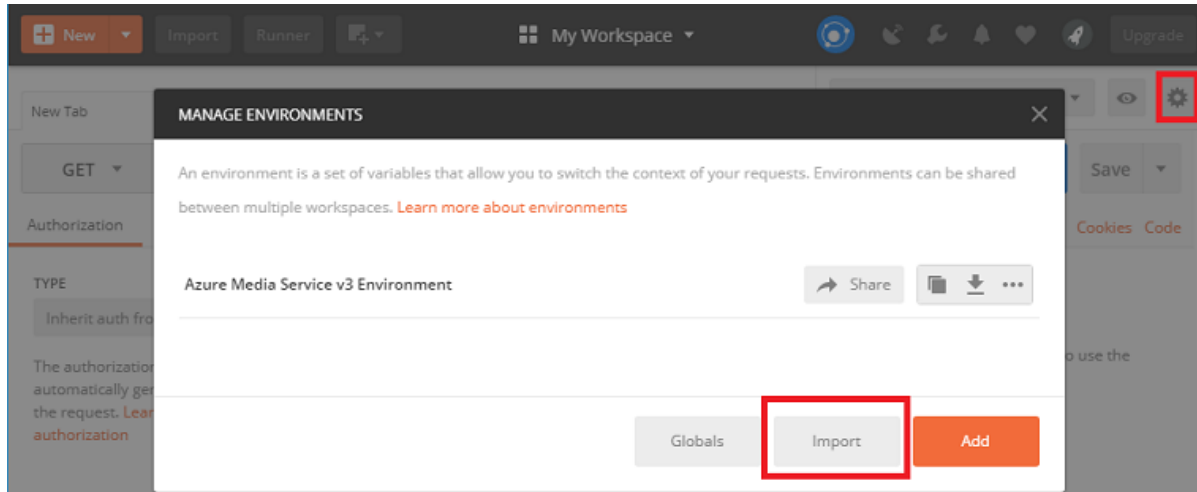
```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup --xml
```

Configure Postman

This section configures the Postman.

Configure the environment

1. Open the **Postman**.
2. On the right of the screen, select the **Manage environment** option.



3. From the **Manage environment** dialog, click **Import**.
4. Browse to the `Azure Media Service v3 Environment.postman_environment.json` file that was downloaded when you cloned `https://github.com/Azure-Samples/media-services-v3-rest-postman.git`.
5. The **Azure Media Service v3 Environment** environment is added.

NOTE

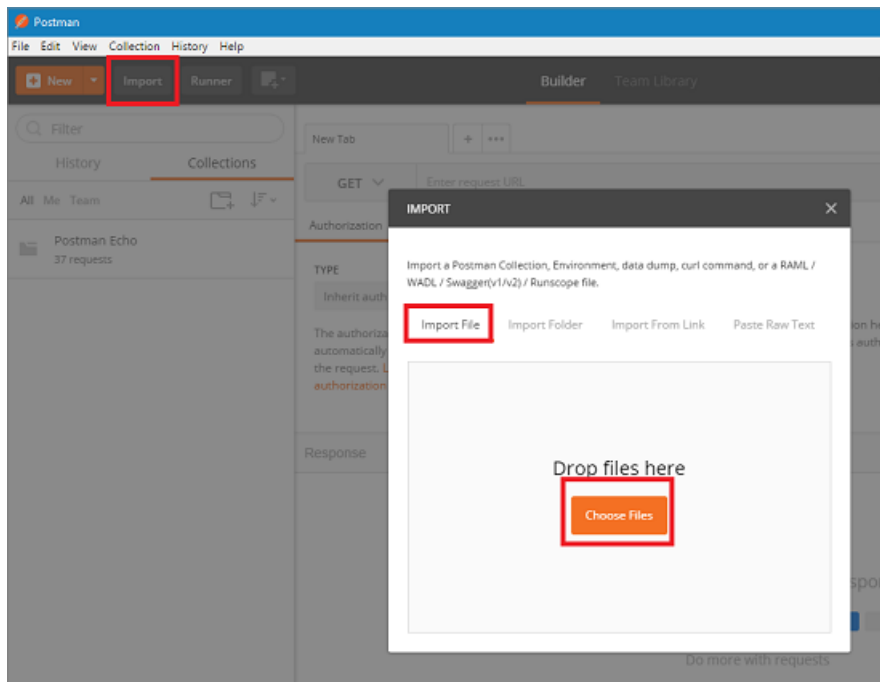
Update access variables with values you got from the **Access the Media Services API** section above.

6. Double-click on the selected file and enter values that you got by following the [accessing API](#) steps.
7. Close the dialog.
8. Select the **Azure Media Service v3 Environment** environment from the dropdown.



Configure the collection

1. Click **Import** to import the collection file.
2. Browse to the `Media Services v3.postman_collection.json` file that was downloaded when you cloned `https://github.com/Azure-Samples/media-services-v3-rest-postman.git`.
3. Choose the **Media Services v3.postman_collection.json** file.



Send requests using Postman

In this section, we send requests that are relevant to encoding and creating URLs so you can stream your file. Specifically, the following requests are sent:

1. Get Azure AD Token for Service Principal Authentication
2. Create an output asset
3. Create a transform
4. Create a job
5. Create a streaming locator
6. List paths of the streaming locator

NOTE

This tutorial assumes you are creating all resources with unique names.

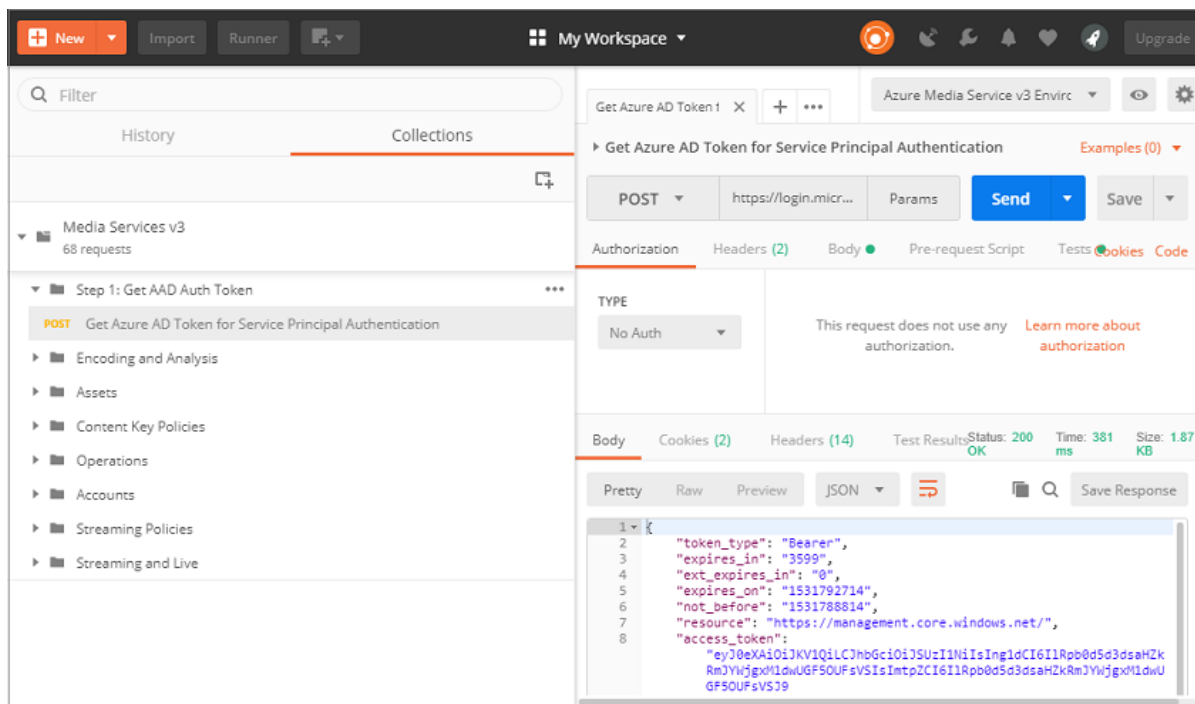
Get Azure AD Token

1. In the left window of the Postman, select "Step 1: Get AAD Auth token".
2. Then, select "Get Azure AD Token for Service Principal Authentication".
3. Press **Send**.

The following **POST** operation is sent.

```
https://login.microsoftonline.com/:tenantId/oauth2/token
```

4. The response comes back with the token and sets the "AccessToken" environment variable to the token value. To see the code that sets "AccessToken", click on the **Tests** tab.



Create an output asset

The output [Asset](#) stores the result of your encoding job.

1. In the left window of the Postman, select "Assets".
2. Then, select "Create or update an Asset".
3. Press **Send**.

- The following **PUT** operation is sent:

```
https://management.azure.com/subscriptions/:subscriptionId/resourceGroups/:resourceGroupName/providers/Microsoft.Media/mediaServices/:accountName/assets/:assetName?api-version={{api-version}}
```

- The operation has the following body:

```
{
  "properties": {
    "description": "My Asset",
    "alternateId" : "some GUID"
  }
}
```

Create a transform

When encoding or processing content in Media Services, it is a common pattern to set up the encoding settings as a recipe. You would then submit a **Job** to apply that recipe to a video. By submitting new Jobs for each new video, you are applying that recipe to all the videos in your library. A recipe in Media Services is called as a **Transform**. For more information, see [Transforms and jobs](#). The sample described in this tutorial defines a recipe that encodes the video in order to stream it to a variety of iOS and Android devices.

When creating a new [Transform](#) instance, you need to specify what you want it to produce as an output. The required parameter is a **TransformOutput** object. Each **TransformOutput** contains a **Preset**. **Preset** describes the step-by-step instructions of video and/or audio processing operations that are to be used to generate the desired **TransformOutput**. The sample described in this article uses a built-in Preset called **AdaptiveStreaming**. The Preset encodes the input video into an auto-generated bitrate ladder (bitrate-resolution pairs) based on the input resolution and bitrate, and produces ISO MP4 files with H.264 video and AAC audio corresponding to each bitrate-resolution pair. For information about this Preset, see [auto-generating bitrate ladder](#).

You can use a built-in `EncoderNamedPreset` or use custom presets.

NOTE

When creating a [Transform](#), you should first check if one already exists using the **Get** method. This tutorial assumes you are creating the transform with a unique name.

1. In the left window of the Postman, select "Encoding and Analysis".
2. Then, select "Create Transform".
3. Press **Send**.

- The following **PUT** operation is sent.

```
https://management.azure.com/subscriptions/:subscriptionId/resourceGroups/:resourceGroupName/providers/Microsoft.Media/mediaServices/:accountName/transforms/:transformName?api-version={{api-version}}
```

- The operation has the following body:

```
{
  "properties": {
    "description": "Basic Transform using an Adaptive Streaming encoding preset from the library of built-in Standard Encoder presets",
    "outputs": [
      {
        "onError": "StopProcessingJob",
        "relativePriority": "Normal",
        "preset": {
          "@odata.type": "#Microsoft.Media.BuiltInStandardEncoderPreset",
          "presetName": "AdaptiveStreaming"
        }
      }
    ]
  }
}
```

Create a job

A **Job** is the actual request to Media Services to apply the created **Transform** to a given input video or audio content. The **Job** specifies information like the location of the input video, and the location for the output.

In this example, the job's input is based on an HTTPS URL ("<https://nimbuscdn-nimbuspm.streaming.mediaservices.windows.net/2b533311-b215-4409-80af-529c3e853622/>").

1. In the left window of the Postman, select "Encoding and Analysis".
2. Then, select "Create or Update Job".
3. Press **Send**.

- The following **PUT** operation is sent.

```
https://management.azure.com/subscriptions/:subscriptionId/resourceGroups/:resourceGroupName/providers/Microsoft.Media/mediaServices/:accountName/transforms/:transformName/jobs/:jobName?api-version={{api-version}}
```

- The operation has the following body:

```
{
  "properties": {
    "input": {
      "@odata.type": "#Microsoft.Media.JobInputHttp",
      "baseUri": "https://nimbuscdn-nimbuspm.streaming.mediaservices.windows.net/2b533311-b215-4409-80af-529c3e853622/",
      "files": [
        "Ignite-short.mp4"
      ]
    },
    "outputs": [
      {
        "@odata.type": "#Microsoft.Media.JobOutputAsset",
        "assetName": "testAsset1"
      }
    ]
  }
}
```

The job takes some time to complete and when it does you want to be notified. To see the progress of the job, we recommend using Event Grid. It is designed for high availability, consistent performance, and dynamic scale. With Event Grid, your apps can listen for and react to events from virtually all Azure services, as well as custom sources. Simple, HTTP-based reactive event handling helps you build efficient solutions through intelligent filtering and routing of events. See [Route events to a custom web endpoint](#).

The **Job** usually goes through the following states: **Scheduled**, **Queued**, **Processing**, **Finished** (the final state). If the job has encountered an error, you get the **Error** state. If the job is in the process of being canceled, you get **Canceling** and **Canceled** when it is done.

Create a streaming locator

After the encoding job is complete, the next step is to make the video in the output Asset available to clients for playback. You can accomplish this in two steps: first, create a [StreamingLocator](#), and second, build the streaming URLs that clients can use.

The process of creating a **StreamingLocator** is called publishing. By default, the **StreamingLocator** is valid immediately after you make the API calls, and lasts until it is deleted, unless you configure the optional start and end times.

When creating a [StreamingLocator](#), you will need to specify the desired **StreamingPolicyName**. In this example, you will be streaming in-the-clear (or non-encrypted) content, so the predefined clear streaming policy (**PredefinedStreamingPolicy.ClearStreamingOnly**) is used.

IMPORTANT

When using a custom [StreamingPolicy](#), you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed.

Your Media Service account has a quota for the number of StreamingPolicy entries. You should not be creating a new StreamingPolicy for each StreamingLocator.

1. In the left window of the Postman, select "Streaming Policies".
2. Then, select "Create a Streaming Locator".
3. Press **Send**.

- The following **PUT** operation is sent.

```
https://management.azure.com/subscriptions/:subscriptionId/resourceGroups/:resourceGroupName/providers/Microsoft.Media/mediaServices/:accountName/streamingPolicies/:streamingPolicyName?api-version={{api-version}}
```

- The operation has the following body:

```
{
  "properties": {
    "assetName": "{{assetName}}",
    "streamingPolicyName": "{{streamingPolicyName}}"
  }
}
```

List paths and build streaming URLs

List paths

Now that the [StreamingLocator](#) has been created, you can get the streaming URLs

1. In the left window of the Postman, select "Streaming Policies".
2. Then, select "List Paths".
3. Press **Send**.

- The following **POST** operation is sent.

```
https://management.azure.com/subscriptions/:subscriptionId/resourceGroups/:resourceGroupName/providers/Microsoft.Media/mediaServices/:accountName/streamingLocators/:streamingLocatorName/listPaths?api-version={{api-version}}
```

- The operation has no body:
4. Note one of the paths you want to use for streaming, you will use it in the next section. In this case, the following paths were returned:

```
"streamingPaths": [
  {
    "streamingProtocol": "Hls",
    "encryptionScheme": "NoEncryption",
    "paths": [
      "/cdb80234-1d94-42a9-b056-0eefa78e5c63/ignite-short.ism/manifest(format=m3u8-aapl)"
    ]
  },
  {
    "streamingProtocol": "Dash",
    "encryptionScheme": "NoEncryption",
    "paths": [
      "/cdb80234-1d94-42a9-b056-0eefa78e5c63/ignite-short.ism/manifest(format=mpd-time-csf)"
    ]
  },
  {
    "streamingProtocol": "SmoothStreaming",
    "encryptionScheme": "NoEncryption",
    "paths": [
      "/cdb80234-1d94-42a9-b056-0eefa78e5c63/ignite-short.ism/manifest"
    ]
  }
]
```

Build the streaming URLs

In this section, let's build an HLS streaming URL. URLs consist of the following values:

1. The protocol over which data is sent. In this case "https".

NOTE

If a player is hosted on an https site, make sure to update the URL to "https".

2. StreamingEndpoint's hostname. In this case, the name is "amsaccount-usw22.streaming.media.azure.net".

To get the hostname, you can use the following GET operation:

```
https://management.azure.com/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/amsResourceGroup/providers/Microsoft.Media/mediaservices/amsaccount/streamingEndpoints/default?api-version={{api-version}}
```

3. A path that you got in the previous (List paths) section.

As a result, the following HLS URL was built

```
https://amsaccount-usw22.streaming.media.azure.net/cdb80234-1d94-42a9-b056-0eefa78e5c63/ignite-short.ism/manifest(format=m3u8-aapl)
```

Test the streaming URL

NOTE

Make sure the streaming endpoint from which you want to stream is running.

To test the stream, this article uses Azure Media Player.

1. Open a web browser and navigate to <https://aka.ms/azuremediaplayer/>.
2. In the **URL:** box, paste the URL you built.
3. Press **Update Player**.

Azure Media Player can be used for testing but should not be used in a production environment.

Clean up resources in your Media Services account

Generally, you should clean up everything except objects that you are planning to reuse (typically, you will reuse Transforms, and you will persist StreamingLocators, etc.). If you want for your account to be clean after experimenting, you should delete the resources that you do not plan to reuse.

To delete a resource, select "Delete ..." operation under whichever resource you want to delete.

Clean up resources

If you no longer need any of the resources in your resource group, including the Media Services and storage accounts you created for this tutorial, delete the resource group you created earlier.

Execute the following CLI command:

```
az group delete --name amsResourceGroup
```

Next steps

Now that you know how to upload, encode, and stream your video, see the following article:

[Analyze videos](#)

Tutorial: Upload, encode, and stream videos using APIs

11/26/2018 • 13 minutes to read • [Edit Online](#)

Azure Media Services enables you to encode your media files into formats that can be played on a wide variety of browsers and devices. For example, you might want to stream your content in Apple's HLS or MPEG DASH formats. Before streaming, you should encode your high-quality digital media file. For encoding guidance, see [Encoding concept](#). This tutorial uploads a local video file and encodes the uploaded file. You can also encode content that you make accessible via a HTTPS URL. For more information, see [Create a job input from an HTTP\(s\) URL](#).

The screenshot displays the Azure Media Player interface. The top navigation bar includes the 'Azure Media Player' logo and the Microsoft logo. Below this is an orange banner with the text 'Your content. Multiple devices. One player.' The main content area is divided into two sections. On the left, the 'Getting Started' sidebar provides instructions on how to use the player, including links to documentation and samples, a list of steps to follow, a 'get share url' button, and a 'start your free trial' button. Below this sidebar is the 'Chosen Player Options' section, which lists the following details: Format: MP4, Tech: Native HTML5 (Type 1), Protection: None/Unknown, and AMP Version: 2.1.5. On the right, a video player is shown, displaying a Microsoft event stage with a large screen showing the Windows logo. Below the video player is the 'Setup' panel, which includes a 'Diagnostics' tab, a 'Code' dropdown, and an 'Adaptive streaming URL from Azure Media Services' section. This section contains a 'Samples' dropdown menu, a 'URL' input field with a sample URL, and an 'Update Player' button. The URL input field also includes a note: 'The URL should contain *.ism/manifest or *.mp4' and an 'Advanced Options' checkbox.

This tutorial shows you how to:

- Access the Media Services API
- Configure the sample app
- Examine the code that uploads, encodes, and streams
- Run the app
- Test the streaming URL
- Clean up resources

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- If you do not have Visual Studio installed, you can get [Visual Studio Community 2017](#).
- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#).

Make sure to remember the values that you used for the resource group name and Media Services account name

Download the sample

Clone a GitHub repository that contains the streaming .NET sample to your machine using the following command:

```
git clone https://github.com/Azure-Samples/media-services-v3-dotnet-tutorials.git
```

The sample is located in the [UploadEncodeAndStreamFiles](#) folder.

Access the Media Services API

To connect to Azure Media Services APIs, you use the Azure AD service principal authentication. The following command creates an Azure AD application and attaches a service principal to the account. You should use the returned values to configure your application.

Before running the script, you can replace the `amsaccount` and `amsResourceGroup` with the names you chose when creating these resources. `amsaccount` is the name of the Azure Media Services account where to attach the service principal.

The following command returns a `json` output:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup
```

This command produces a response similar to this:

```
{
  "AadClientId": "00000000-0000-0000-0000-000000000000",
  "AadEndpoint": "https://login.microsoftonline.com",
  "AadSecret": "00000000-0000-0000-0000-000000000000",
  "AadTenantId": "00000000-0000-0000-0000-000000000000",
  "AccountName": "amsaccount",
  "ArmAadAudience": "https://management.core.windows.net/",
  "ArmEndpoint": "https://management.azure.com/",
  "Region": "West US 2",
  "ResourceGroup": "amsResourceGroup",
  "SubscriptionId": "00000000-0000-0000-0000-000000000000"
}
```

If you would like to get an `xml` in the response, use the following command:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup --xml
```

Examine the code that uploads, encodes, and streams

This section examines functions defined in the [Program.cs](#) file of the *UploadEncodeAndStreamFiles* project.

The sample performs the following actions:

1. Creates a new Transform (first, checks if the specified Transform exists).
2. Creates an output Asset that is used as the encoding Job's output.
3. Create an input Asset and uploads the specified local video file into it. The Asset is used as the Job's input.
4. Submits the encoding Job using the input and output that was created.
5. Checks the Job's status.
6. Creates a StreamingLocator.
7. Builds streaming URLs.

Start using Media Services APIs with .NET SDK

To start using Media Services APIs with .NET, you need to create an **AzureMediaServicesClient** object. To create the object, you need to supply credentials needed for the client to connect to Azure using Azure AD. In the code you cloned at the beginning of the article, the **GetCredentialsAsync** function creates the **ServiceClientCredentials** object based on the credentials supplied in local configuration file.

```
private static async Task<IAzureMediaServicesClient> CreateMediaServicesClientAsync(ConfigWrapper config)
{
    var credentials = await GetCredentialsAsync(config);

    return new AzureMediaServicesClient(config.ArmEndpoint, credentials)
    {
        SubscriptionId = config.SubscriptionId,
    };
}
```

Create an input asset and upload a local file into it

The **CreateInputAsset** function creates a new input [Asset](#) and uploads the specified local video file into it. This Asset is used as the input to your encoding Job. In Media Services v3, the input to a Job can either be an Asset, or it can be content that you make available to your Media Services account via HTTPS URLs. If you want to learn how to encode from a HTTPS URL, see [this](#) article.

In Media Services v3, you use Azure Storage APIs to upload files. The following .NET snippet shows how.

The following function performs these actions:

- Creates an Asset
- Gets a writable [SAS URL](#) to the Asset's [container in storage](#)
- Uploads the file into the container in storage using the SAS URL


```

private static async Task<Asset> CreateInputAssetAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string assetName,
    string fileToUpload)
{
    // In this example, we are assuming that the asset name is unique.
    //
    // If you already have an asset with the desired name, use the Assets.Get method
    // to get the existing asset. In Media Services v3, the Get method on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).

    // Call Media Services API to create an Asset.
    // This method creates a container in storage for the Asset.
    // The files (blobs) associated with the asset will be stored in this container.
    Asset asset = await client.Assets.CreateOrUpdateAsync(resourceGroupName, accountName, assetName, new
Asset());

    // Use Media Services API to get back a response that contains
    // SAS URL for the Asset container into which to upload blobs.
    // That is where you would specify read-write permissions
    // and the expiration time for the SAS URL.
    var response = await client.Assets.ListContainerSasAsync(
        resourceGroupName,
        accountName,
        assetName,
        permissions: AssetContainerPermission.ReadWrite,
        expiryTime: DateTime.UtcNow.AddHours(4).ToUniversalTime());

    var sasUri = new Uri(response.AssetContainerSasUrls.First());

    // Use Storage API to get a reference to the Asset container
    // that was created by calling Asset's CreateOrUpdate method.
    CloudBlobContainer container = new CloudBlobContainer(sasUri);
    var blob = container.GetBlockBlobReference(Path.GetFileName(fileToUpload));

    // Use Storage API to upload the file into the container in storage.
    await blob.UploadFromFileAsync(fileToUpload);

    return asset;
}

```

Create an output asset to store the result of a job

The output [Asset](#) stores the result of your encoding job. The project defines the **DownloadResults** function that downloads the results from this output asset into the "output" folder, so you can see what you got.

```

private static async Task<Asset> CreateOutputAssetAsync(IAzureMediaServicesClient client, string
resourceGroupName, string accountName, string assetName)
{
    // Check if an Asset already exists
    Asset outputAsset = await client.Assets.GetAsync(resourceGroupName, accountName, assetName);
    Asset asset = new Asset();
    string outputAssetName = assetName;

    if (outputAsset != null)
    {
        // Name collision! In order to get the sample to work, let's just go ahead and create a unique asset
name
        // Note that the returned Asset can have a different name than the one specified as an input
parameter.
        // You may want to update this part to throw an Exception instead, and handle name collisions
differently.
        string uniqueness = $"-{Guid.NewGuid().ToString("N")}";
        outputAssetName += uniqueness;

        Console.WriteLine("Warning - found an existing Asset with name = " + assetName);
        Console.WriteLine("Creating an Asset with this name instead: " + outputAssetName);
    }

    return await client.Assets.CreateOrUpdateAsync(resourceGroupName, accountName, outputAssetName, asset);
}

```

Create a Transform and a Job that encodes the uploaded file

When encoding or processing content in Media Services, it is a common pattern to set up the encoding settings as a recipe. You would then submit a **Job** to apply that recipe to a video. By submitting new Jobs for each new video, you are applying that recipe to all the videos in your library. A recipe in Media Services is called as a **Transform**. For more information, see [Transforms and jobs](#). The sample described in this tutorial defines a recipe that encodes the video in order to stream it to a variety of iOS and Android devices.

Transform

When creating a new [Transform](#) instance, you need to specify what you want it to produce as an output. The required parameter is a **TransformOutput** object, as shown in the code below. Each **TransformOutput** contains a **Preset**. **Preset** describes the step-by-step instructions of video and/or audio processing operations that are to be used to generate the desired **TransformOutput**. The sample described in this article uses a built-in Preset called **AdaptiveStreaming**. The Preset encodes the input video into an auto-generated bitrate ladder (bitrate-resolution pairs) based on the input resolution and bitrate, and produces ISO MP4 files with H.264 video and AAC audio corresponding to each bitrate-resolution pair. For information about this Preset, see [auto-generating bitrate ladder](#).

You can use a built-in EncoderNamedPreset or use custom presets. For more information, see [How to customize encoder presets](#).

When creating a [Transform](#), you should first check if one already exists using the **Get** method, as shown in the code that follows. In Media Services v3, **Get** methods on entities return **null** if the entity doesn't exist (a case-insensitive check on the name).

```

private static async Task<Transform> GetOrCreateTransformAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName)
{
    // Does a Transform already exist with the desired name? Assume that an existing Transform with the
    // desired name
    // also uses the same recipe or Preset for processing content.
    Transform transform = await client.Transforms.GetAsync(resourceGroupName, accountName, transformName);

    if (transform == null)
    {
        // You need to specify what you want it to produce as an output
        TransformOutput[] output = new TransformOutput[]
        {
            new TransformOutput
            {
                // The preset for the Transform is set to one of Media Services built-in sample presets.
                // You can customize the encoding settings by changing this to use "StandardEncoderPreset"
                class.
                Preset = new BuiltInStandardEncoderPreset()
                {
                    // This sample uses the built-in encoding preset for Adaptive Bitrate Streaming.
                    PresetName = EncoderNamedPreset.AdaptiveStreaming
                }
            }
        };

        // Create the Transform with the output defined above
        transform = await client.Transforms.CreateOrUpdateAsync(resourceGroupName, accountName,
            transformName, output);
    }

    return transform;
}

```

Job

As mentioned above, the [Transform](#) object is the recipe and a [Job](#) is the actual request to Media Services to apply that **Transform** to a given input video or audio content. The **Job** specifies information like the location of the input video, and the location for the output.

In this example, the input video has been uploaded from your local machine. If you want to learn how to encode from a HTTPS URL, see [this](#) article.

```

private static async Task<Job> SubmitJobAsync(IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string jobName,
    string inputAssetName,
    string outputAssetName)
{
    // Use the name of the created input asset to create the job input.
    JobInput jobInput = new JobInputAsset(assetName: inputAssetName);

    JobOutput[] jobOutputs =
    {
        new JobOutputAsset(outputAssetName),
    };

    // In this example, we are assuming that the job name is unique.
    //
    // If you already have a job with the desired name, use the Jobs.Get method
    // to get the existing job. In Media Services v3, the Get method on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).
    Job job = await client.Jobs.CreateAsync(
        resourceGroupName,
        accountName,
        transformName,
        jobName,
        new Job
        {
            Input = jobInput,
            Outputs = jobOutputs,
        });

    return job;
}

```

Wait for the Job to complete

The job takes some time to complete and when it does you want to be notified. The code sample below shows how to poll the service for the status of the [Job](#). Polling is not a recommended best practice for production applications because of potential latency. Polling can be throttled if overused on an account. Developers should instead use Event Grid.

Event Grid is designed for high availability, consistent performance, and dynamic scale. With Event Grid, your apps can listen for and react to events from virtually all Azure services, as well as custom sources. Simple, HTTP-based reactive event handling helps you build efficient solutions through intelligent filtering and routing of events. See [Route events to a custom web endpoint](#).

The **Job** usually goes through the following states: **Scheduled**, **Queued**, **Processing**, **Finished** (the final state). If the job has encountered an error, you get the **Error** state. If the job is in the process of being canceled, you get **Canceling** and **Canceled** when it is done.

```

private static async Task<Job> WaitForJobToFinishAsync(IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string jobName)
{
    const int SleepIntervalMs = 60 * 1000;

    Job job = null;

    do
    {
        job = await client.Jobs.GetAsync(resourceGroupName, accountName, transformName, jobName);

        Console.WriteLine($"Job is '{job.State}'");
        for (int i = 0; i < job.Outputs.Count; i++)
        {
            JobOutput output = job.Outputs[i];
            Console.WriteLine($"JobOutput[{i}] is '{output.State}'");
            if (output.State == JobState.Processing)
            {
                Console.WriteLine($"Progress: '{output.Progress}'");
            }

            Console.WriteLine();
        }

        if (job.State != JobState.Finished && job.State != JobState.Error && job.State != JobState.Canceled)
        {
            await Task.Delay(SleepIntervalMs);
        }
    }
    while (job.State != JobState.Finished && job.State != JobState.Error && job.State != JobState.Canceled);

    return job;
}

```

Get a StreamingLocator

After the encoding is complete, the next step is to make the video in the output Asset available to clients for playback. You can accomplish this in two steps: first, create a [StreamingLocator](#), and second, build the streaming URLs that clients can use.

The process of creating a **StreamingLocator** is called publishing. By default, the **StreamingLocator** is valid immediately after you make the API calls, and lasts until it is deleted, unless you configure the optional start and end times.

When creating a [StreamingLocator](#), you will need to specify the desired **StreamingPolicyName**. In this example, you will be streaming in-the-clear (or non-encrypted content) so the predefined clear streaming policy (**PredefinedStreamingPolicy.ClearStreamingOnly**) is used.

IMPORTANT

When using a custom [StreamingPolicy](#), you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed. Your Media Service account has a quota for the number of StreamingPolicy entries. You should not be creating a new StreamingPolicy for each StreamingLocator.

The following code assumes that you are calling the function with a unique locatorName.

```
private static async Task<StreamingLocator> CreateStreamingLocatorAsync(
    IAzureMediaServicesClient client,
    string resourceGroup,
    string accountName,
    string assetName,
    string locatorName)
{
    StreamingLocator locator = await client.StreamingLocators.CreateAsync(
        resourceGroup,
        accountName,
        locatorName,
        new StreamingLocator
        {
            AssetName = assetName,
            StreamingPolicyName = PredefinedStreamingPolicy.ClearStreamingOnly
        });

    return locator;
}
```

While the sample in this topic discusses streaming, you can use the same call to create a **StreamingLocator** for delivering video via progressive download.

Get streaming URLs

Now that the **StreamingLocator** has been created, you can get the streaming URLs, as shown in **GetStreamingURLs**. To build a URL, you need to concatenate the **StreamingEndpoint** host name and the **StreamingLocator** path. In this sample, the *default* **StreamingEndpoint** is used. When you first create a Media Service account, this *default* **StreamingEndpoint** will be in a stopped state, so you need to call **Start**.

NOTE

In this method, you need the **locatorName** that was used when creating the **StreamingLocator** for the output Asset.

```

private static async Task<IList<string>> GetStreamingUrlsAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    String locatorName)
{
    const string DefaultStreamingEndpointName = "default";

    IList<string> streamingUrls = new List<string>();

    StreamingEndpoint streamingEndpoint = await client.StreamingEndpoints.GetAsync(resourceGroupName,
accountName, DefaultStreamingEndpointName);

    if (streamingEndpoint != null)
    {
        if (streamingEndpoint.ResourceState != StreamingEndpointResourceState.Running)
        {
            await client.StreamingEndpoints.StartAsync(resourceGroupName, accountName,
DefaultStreamingEndpointName);
        }
    }

    ListPathsResponse paths = await client.StreamingLocators.ListPathsAsync(resourceGroupName, accountName,
locatorName);

    foreach (StreamingPath path in paths.StreamingPaths)
    {
        UriBuilder uriBuilder = new UriBuilder();
        uriBuilder.Scheme = "https";
        uriBuilder.Host = streamingEndpoint.HostName;

        uriBuilder.Path = path.Paths[0];
        streamingUrls.Add(uriBuilder.ToString());
    }

    return streamingUrls;
}

```

Clean up resources in your Media Services account

Generally, you should clean up everything except objects that you are planning to reuse (typically, you will reuse Transforms, and you will persist StreamingLocators, etc.). If you want for your account to be clean after experimenting, you should delete the resources that you do not plan to reuse. For example, the following code deletes Jobs.

```

private static async Task CleanUpAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName)
{
    var jobs = await client.Jobs.ListAsync(resourceGroupName, accountName, transformName);
    foreach (var job in jobs)
    {
        await client.Jobs.DeleteAsync(resourceGroupName, accountName, transformName, job.Name);
    }

    var assets = await client.Assets.ListAsync(resourceGroupName, accountName);
    foreach (var asset in assets)
    {
        await client.Assets.DeleteAsync(resourceGroupName, accountName, asset.Name);
    }
}

```

Run the sample app

1. Press Ctrl+F5 to run the *EncodeAndStreamFiles* application.
2. Copy one of the streaming URLs from the console.

This example displays URLs that can be used to play back the video using different protocols:

```
C:\WINDOWS\system32\cmd.exe
Job is Queued.
  JobOutput[0] is Queued.
Job is Processing.
  JobOutput[0] is Processing. Progress: 0
Job is Processing.
  JobOutput[0] is Processing. Progress: 24
Job is Processing.
  JobOutput[0] is Processing. Progress: 49
Job is Processing.
  JobOutput[0] is Processing. Progress: 76
Downloading results to D:\AMS\output\d894fd92-a050-45e7-871a-34a230745d5c-output.
Download complete.
http://amstest.streaming.mediaservices.windows.net/d4acbbdf-657a-4954-b7ce-d7c85c8b4796/ignite-short.ism/manifest(format=m3u8-aapl)
http://amstest.streaming.mediaservices.windows.net/d4acbbdf-657a-4954-b7ce-d7c85c8b4796/ignite-short.ism/manifest(format=mpd-time-csf)
http://amstest.streaming.mediaservices.windows.net/d4acbbdf-657a-4954-b7ce-d7c85c8b4796/ignite-short.ism/manifest
Press any key to continue . . .
```

Test the streaming URL

To test the stream, this article uses Azure Media Player.

NOTE

If a player is hosted on an https site, make sure to update the URL to "https".

1. Open a web browser and navigate to <https://aka.ms/azuremediaplayer/>.
2. In the **URL:** box, paste one of the streaming URL values you got when you ran the application.
3. Press **Update Player**.

Azure Media Player can be used for testing but should not be used in a production environment.

Clean up resources

If you no longer need any of the resources in your resource group, including the Media Services and storage accounts you created for this tutorial, delete the resource group you created earlier.

Execute the following CLI command:

```
az group delete --name amsResourceGroup
```

Multithreading

The Azure Media Services v3 SDKs are not thread-safe. When developing a multi-threaded application, you should generate and use a new *AzureMediaServicesClient* object per thread.

Next steps

Now that you know how to upload, encode, and stream your video, see the following article:

[Analyze videos](#)

Tutorial: Stream live with Media Services v3 using APIs

11/13/2018 • 10 minutes to read • [Edit Online](#)

In Azure Media Services, [LiveEvents](#) are responsible for processing live streaming content. A LiveEvent provides an input endpoint (ingest URL) that you then provide to a live encoder. The LiveEvent receives live input streams from the live encoder and makes it available for streaming through one or more [StreamingEndpoints](#). LiveEvents also provide a preview endpoint (preview URL) that you use to preview and validate your stream before further processing and delivery. This tutorial shows how to use .NET Core to create a **pass-through** type of a live event.

NOTE

Make sure to review [Live streaming with Media Services v3](#) before proceeding.

The tutorial shows you how to:

- Access the Media Services API
- Configure the sample app
- Examine the code that performs live streaming
- Watch the event with [Azure Media Player](#) at <http://ampdemo.azureedge.net>
- Clean up resources

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

The following are required to complete the tutorial.

- Install Visual Studio Code or Visual Studio.
- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#).

Make sure to remember the values that you used for the resource group name and Media Services account name

- A camera or a device (like laptop) that is used to broadcast an event.
- An on-premises live encoder that converts signals from the camera to streams that are sent to the Media Services live streaming service. The stream has to be in **RTMP** or **Smooth Streaming** format.

Download the sample

Clone a GitHub repository that contains the streaming .NET sample to your machine using the following command:

```
git clone https://github.com/Azure-Samples/media-services-v3-dotnet-core-tutorials.git
```

The live streaming sample is located in the [Live](#) folder.

IMPORTANT

This sample uses unique suffix for each resource. If you cancel the debugging or terminate the app without running it through, you will end up with multiple LiveEvents in your account.

Make sure to stop the running LiveEvents. Otherwise, you will be **billed**!

Access the Media Services API

To connect to Azure Media Services APIs, you use the Azure AD service principal authentication. The following command creates an Azure AD application and attaches a service principal to the account. You should use the returned values to configure your application.

Before running the script, you can replace the `amsaccount` and `amsResourceGroup` with the names you chose when creating these resources. `amsaccount` is the name of the Azure Media Services account where to attach the service principal.

The following command returns a `json` output:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup
```

This command produces a response similar to this:

```
{
  "AadClientId": "00000000-0000-0000-0000-000000000000",
  "AadEndpoint": "https://login.microsoftonline.com",
  "AadSecret": "00000000-0000-0000-0000-000000000000",
  "AadTenantId": "00000000-0000-0000-0000-000000000000",
  "AccountName": "amsaccount",
  "ArmAadAudience": "https://management.core.windows.net/",
  "ArmEndpoint": "https://management.azure.com/",
  "Region": "West US 2",
  "ResourceGroup": "amsResourceGroup",
  "SubscriptionId": "00000000-0000-0000-0000-000000000000"
}
```

If you would like to get an `xml` in the response, use the following command:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup --xml
```

Examine the code that performs live streaming

This section examines functions defined in the [Program.cs](#) file of the *MediaV3LiveApp* project.

The sample creates a unique suffix for each resource so that we don't have name collisions if you run the sample multiple times without cleaning up.

IMPORTANT

This sample uses unique suffix for each resource. If you cancel the debugging or terminate the app without running it through, you will end up with multiple LiveEvents in your account.

Make sure to stop the running LiveEvents. Otherwise, you will be **billed**!

Start using Media Services APIs with .NET SDK

To start using Media Services APIs with .NET, you need to create an **AzureMediaServicesClient** object. To create the object, you need to supply credentials needed for the client to connect to Azure using Azure AD. In the code you cloned at the beginning of the article, the **GetCredentialsAsync** function creates the **ServiceClientCredentials** object based on the credentials supplied in local configuration file.

```
private static async Task<IAzureMediaServicesClient> CreateMediaServicesClientAsync(ConfigWrapper config)
{
    var credentials = await GetCredentialsAsync(config);

    return new AzureMediaServicesClient(config.ArmEndpoint, credentials)
    {
        SubscriptionId = config.SubscriptionId,
    };
}
```

Create a live event

This section shows how to create a **pass-through** type of LiveEvent (LiveEventEncodingType set to None). If you want to create a LiveEvent that is enabled for live encoding set LiveEventEncodingType to Basic.

Some other things that you might want to specify when creating the live event are:

- Media Services location
- The streaming protocol for the Live Event (currently, the RTMP and Smooth Streaming protocols are supported)

You cannot change the protocol option while the LiveEvent or its associated LiveOutputs are running. If you require different protocols, you should create separate LiveEvent for each streaming protocol.

- IP restrictions on the ingest and preview. You can define the IP addresses that are allowed to ingest a video to this LiveEvent. Allowed IP addresses can be specified as either a single IP address (for example '10.0.0.1'), an IP range using an IP address and a CIDR subnet mask (for example, '10.0.0.1/22'), or an IP range using an IP address and a dotted decimal subnet mask (for example, '10.0.0.1(255.255.252.0)').

If no IP addresses are specified and there is no rule definition, then no IP address will be allowed. To allow any IP address, create a rule and set 0.0.0.0/0.

When creating the event, you can specify to auto start it.

```
Console.WriteLine($"Creating a live event named {liveEventName}");
Console.WriteLine();

// Create the LiveEvent input IP access control
LiveEventInputAccessControl liveEventInputAccess = new LiveEventInputAccessControl
{
    Ip = new IPAccessControl(
        allow: new IPRange[]
        {
            new IPRange (
                name: "AllowAll",
                address: "0.0.0.0",
                subnetPrefixLength: 0
            )
        }
    )
}
```

```

    )
    }
)

};

// Create the LiveEvent Preview IP access control
LiveEventPreview liveEventPreview = new LiveEventPreview
{
    AccessControl = new LiveEventPreviewAccessControl(
        ip: new IPAccessControl(
            allow: new IPRange[]
            {
                new IPRange (
                    name: "AllowAll",
                    address: "0.0.0.0",
                    subnetPrefixLength: 0
                )
            }
        )
    )
}

};

// To get the same ingest URL for the same LiveEvent name:
// 1. Set vanityUrl to true so you have ingest like:
//      rtmps://liveevent-hevc12-eventgridmediaservice-
//      usw22.channel.media.azure.net:2935/live/522f9b27dd2d4b26aeb9ef8ab96c5c77
// 2. Set accessToken to a desired GUID string (with or without hyphen)

// The following operation can sometimes take awhile. Be patient.
LiveEvent liveEvent = new LiveEvent(
    location: mediaService.Location,
    description:"Sample LiveEvent for testing",
    vanityUrl:false,
    encoding: new LiveEventEncoding(
        // Set this to Basic to enable a transcoding LiveEvent, and None to enable a pass-through
LiveEvent
        encodingType:LiveEventEncodingType.None,
        presetName:null
    ),
    input: new LiveEventInput(LiveEventInputProtocol.RTMP, liveEventInputAccess),
    preview: liveEventPreview,
    streamOptions: new List<StreamOptionsFlag?>()
    {
        // Set this to Default or Low Latency
        // When using Low Latency mode, you must configure the Azure Media Player to use the
        // quick start heuristic profile or you won't notice the change.
        // In the AMP player client side JS options, set - heuristicProfile: "Low Latency Heuristic Profile".
        // To use low latency optimally, you should tune your encoder settings down to 1 second GOP size
        // instead of 2 seconds.
        StreamOptionsFlag.LowLatency
    }
);

Console.WriteLine($"Creating the LiveEvent, be patient this can take time...");
liveEvent = client.LiveEvents.Create(config.ResourceGroup, config.AccountName, liveEventName, liveEvent,
autoStart:true);

```

Get ingest URLs

Once the channel is created, you can get ingest URLs that you will provide to the live encoder. The encoder uses these URLs to input a live stream.

```
string ingestUrl = liveEvent.Input.Endpoints.First().Url;
Console.WriteLine($"The ingest url to configure the on premise encoder with is:");
Console.WriteLine($"{ingestUrl}");
Console.WriteLine();
```

Get the preview URL

Use the previewEndpoint to preview and verify that the input from the encoder is actually being received.

IMPORTANT

Make sure that the video is flowing to the Preview URL before continuing!

```
string previewEndpoint = liveEvent.Preview.Endpoints.First().Url;
Console.WriteLine($"The preview url is:");
Console.WriteLine($"{previewEndpoint}");
Console.WriteLine();

Console.WriteLine($"Open the live preview in your browser and use the Azure Media Player to monitor the
preview playback:");
Console.WriteLine($"https://ampdemo.azureedge.net/?url={previewEndpoint}&heuristicprofile=lowlatency");
Console.WriteLine();
```

Create and manage LiveEvents and LiveOutputs

Once you have the stream flowing into the LiveEvent, you can begin the streaming event by creating an Asset, LiveOutput, and StreamingLocator. This will archive the stream and make it available to viewers through the StreamingEndpoint.

Create an Asset

```
string assetName = "archiveAsset" + uniqueness;
Console.WriteLine($"Creating an asset named {assetName}");
Console.WriteLine();
Asset asset = client.Assets.CreateOrUpdate(config.ResourceGroup, config.AccountName, assetName, new Asset());
```

Create an Asset for the LiveOutput to use.

Create a LiveOutput

```
string manifestName = "output";
string liveOutputName = "liveOutput" + uniqueness;
Console.WriteLine($"Creating a live output named {liveOutputName}");
Console.WriteLine();

LiveOutput liveOutput = new LiveOutput(assetName: asset.Name, manifestName: manifestName, archiveWindowLength:
    TimeSpan.FromMinutes(10));
liveOutput = client.LiveOutputs.Create(config.ResourceGroup, config.AccountName, liveEventName,
    liveOutputName, liveOutput);
```

Create a StreamingLocator

NOTE

When your Media Services account is created a **default** streaming endpoint is added to your account in the **Stopped** state. To start streaming your content and take advantage of dynamic packaging and dynamic encryption, the streaming endpoint from which you want to stream content has to be in the **Running** state.

```

StreamingLocator locator = new StreamingLocator(assetName: asset.Name, streamingPolicyName:
PredefinedStreamingPolicy.ClearStreamingOnly);
locator = client.StreamingLocators.Create(config.ResourceGroup, config.AccountName, streamingLocatorName,
locator);

// Get the default Streaming Endpoint on the account
StreamingEndpoint streamingEndpoint = client.StreamingEndpoints.Get(config.ResourceGroup, config.AccountName,
"default");

// If it's not running, Start it.
if (streamingEndpoint.ResourceState != StreamingEndpointResourceState.Running)
{
    Console.WriteLine("Streaming Endpoint was Stopped, restarting now..");
    client.StreamingEndpoints.Start(config.ResourceGroup, config.AccountName, "default");
}

```

```

// Get the url to stream the output
ListPathsResponse paths = await client.StreamingLocators.ListPathsAsync(resourceGroupName, accountName,
locatorName);

foreach (StreamingPath path in paths.StreamingPaths)
{
    UriBuilder uriBuilder = new UriBuilder();
    uriBuilder.Scheme = "https";
    uriBuilder.Host = streamingEndpoint.HostName;

    uriBuilder.Path = path.Paths[0];
    // Get the URL from the uriBuilder: uriBuilder.ToString()
}

```

Cleaning up resources in your Media Services account

If you are done streaming events and want to clean up the resources provisioned earlier, follow the following procedure.

- Stop pushing the stream from the encoder.
- Stop the LiveEvent. Once the LiveEvent is stopped, it will not incur any charges. When you need to start it again, it will have the same ingest URL so you won't need to reconfigure your encoder.
- You can stop your StreamingEndpoint, unless you want to continue to provide the archive of your live event as an on-demand stream. If the LiveEvent is in stopped state, it will not incur any charges.

```

private static void CleanupLiveEventAndOutput(IAzureMediaServicesClient client, string resourceGroup, string
accountName, string liveEventName, string liveOutputName)
{
    // Delete the LiveOutput
    client.LiveOutputs.Delete(resourceGroup, accountName, liveEventName, liveOutputName);

    // Stop and delete the LiveEvent
    client.LiveEvents.Stop(resourceGroup, accountName, liveEventName);
    client.LiveEvents.Delete(resourceGroup, accountName, liveEventName);
}

```

```

private static void CleanupLocatorAssetAndStreamingEndpoint(IAzureMediaServicesClient client, string
resourceGroup, string accountName, string streamingLocatorName, string assetName)
{
    // Delete the Streaming Locator
    client.StreamingLocators.Delete(resourceGroup, accountName, streamingLocatorName);

    // Delete the Archive Asset
    client.Assets.Delete(resourceGroup, accountName, assetName);

    // Stop and delete the StreamingEndpoint
    // client.StreamingEndpoints.Stop(resourceGroup, accountName, endpointName);
    // client.StreamingEndpoints.Delete(resourceGroup, accountName, endpointName);

}

```

```

private static void CleanupAccount(IAzureMediaServicesClient client, string resourceGroup, string accountName)
{
    try{

        Console.WriteLine("Cleaning up the resources used, stopping the LiveEvent. This can take a few minutes
to complete.");
        Console.WriteLine();

        var events = client.LiveEvents.List(resourceGroup, accountName);

        foreach (LiveEvent l in events)
        {
            if (l.Name == liveEventName){
                var outputs = client.LiveOutputs.List(resourceGroup, accountName, l.Name);

                foreach (LiveOutput o in outputs)
                {
                    client.LiveOutputs.Delete(resourceGroup, accountName, l.Name, o.Name);
                    Console.WriteLine($"LiveOutput: {o.Name} deleted from LiveEvent {l.Name}. The archived
Asset and Streaming URLs are still retained for on-demand viewing.");
                }

                if (l.ResourceState == LiveEventResourceState.Running){
                    client.LiveEvents.Stop(resourceGroup, accountName, l.Name);
                    Console.WriteLine($"LiveEvent: {l.Name} Stopped.");
                    client.LiveEvents.Delete(resourceGroup, accountName, l.Name);
                    Console.WriteLine($"LiveEvent: {l.Name} Deleted.");
                    Console.WriteLine();
                }
            }
        }

    }
    catch(ApiErrorException e)
    {
        Console.WriteLine("Hit ApiErrorException");
        Console.WriteLine($"\\tCode: {e.Body.Error.Code}");
        Console.WriteLine($"\\tCode: {e.Body.Error.Message}");
        Console.WriteLine();
    }
}

```

Watch the event

To watch the event, copy the streaming URL that you got when you ran code described in [Create a StreamingLocator](#) and use a player of your choice. You can use [Azure Media Player](#) to test your stream at <http://ampdemo.azureedge.net>.

Live event automatically converts events to on-demand content when stopped. Even after you stop and delete the event, the users would be able to stream your archived content as a video on demand, for as long as you do not delete the asset. An asset cannot be deleted if it is used by an event; the event must be deleted first.

Clean up resources

If you no longer need any of the resources in your resource group, including the Media Services and storage accounts you created for this tutorial, delete the resource group you created earlier.

Execute the following CLI command:

```
az group delete --name amsResourceGroup
```

IMPORTANT

Leaving the LiveEvent running incurs billing costs. Be aware, if the project/program crashes or is closed out for any reason, it could leave the LiveEvent running in a billing state.

Next steps

[Stream files](#)

Tutorial: Analyze videos with Media Services v3 using APIs

12/10/2018 • 11 minutes to read • [Edit Online](#)

This tutorial shows you how to analyze videos with Azure Media Services. There are many scenarios in which you might want to gain deep insights into recorded videos or audio content. For example, to achieve higher customer satisfaction, organizations can run speech-to-text processing to convert customer support recordings into a searchable catalog, with indexes and dashboards. Then, they can obtain insights into their business such as a list of common complaints, sources of such complaints, and other useful information.

This tutorial shows you how to:

- Create a Media Services account
- Access the Media Services API
- Configure the sample app
- Examine the code that analyzes the specified video
- Run the app
- Examine the output
- Clean up resources

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- If you do not have Visual Studio installed, you can get [Visual Studio Community 2017](#).
- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#).

Make sure to remember the values that you used for the resource group name and Media Services account name.

Download the sample

Clone a GitHub repository that contains the .NET sample to your machine using the following command:

```
git clone https://github.com/Azure-Samples/media-services-v3-dotnet-tutorials.git
```

The sample is located in the [AnalyzeVideos](#) folder.

Access the Media Services API

To connect to Azure Media Services APIs, you use the Azure AD service principal authentication. The following command creates an Azure AD application and attaches a service principal to the account. You should use the returned values to configure your application.

Before running the script, you can replace the `amsaccount` and `amsResourceGroup` with the names you chose when creating these resources. `amsaccount` is the name of the Azure Media Services account where to attach the service principal.

The following command returns a `json` output:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup
```

This command produces a response similar to this:

```
{
  "AadClientId": "00000000-0000-0000-0000-000000000000",
  "AadEndpoint": "https://login.microsoftonline.com",
  "AadSecret": "00000000-0000-0000-0000-000000000000",
  "AadTenantId": "00000000-0000-0000-0000-000000000000",
  "AccountName": "amsaccount",
  "ArmAadAudience": "https://management.core.windows.net/",
  "ArmEndpoint": "https://management.azure.com/",
  "Region": "West US 2",
  "ResourceGroup": "amsResourceGroup",
  "SubscriptionId": "00000000-0000-0000-0000-000000000000"
}
```

If you would like to get an `xml` in the response, use the following command:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup --xml
```

Examine the code that analyzes the specified video

This section examines functions defined in the `Program.cs` file of the *AnalyzeVideos* project.

The sample performs the following actions:

1. Creates a transform and a job that analyzes your video.
2. Creates an input asset and uploads the video into it. The asset is used as the job's input.
3. Creates an output asset that stores the job's output.
4. Submits the job.
5. Checks the job's status.
6. Downloads the files that resulted from running the job.

NOTE

When using a Video or Audio Analyzer presets, use the Azure portal to set your account to have 10 S3 Media Reserved Units. For more information, see [Scale media processing](#).

Start using Media Services APIs with .NET SDK

To start using Media Services APIs with .NET, you need to create an **AzureMediaServicesClient** object. To create the object, you need to supply credentials needed for the client to connect to Azure using Azure AD. In the code you cloned at the beginning of the article, the **GetCredentialsAsync** function creates the **ServiceClientCredentials** object based on the credentials supplied in local configuration file.

```
private static async Task<IAzureMediaServicesClient> CreateMediaServicesClientAsync(ConfigWrapper config)
{
    var credentials = await GetCredentialsAsync(config);

    return new AzureMediaServicesClient(config.ArmEndpoint, credentials)
    {
        SubscriptionId = config.SubscriptionId,
    };
}
```

Create an input asset and upload a local file into it

The **CreateInputAsset** function creates a new input [Asset](#) and uploads the specified local video file into it. This Asset is used as the input to your encoding Job. In Media Services v3, the input to a Job can either be an Asset, or it can be content that you make available to your Media Services account via HTTPS URLs. If you want to learn how to encode from a HTTPS URL, see [this](#) article.

In Media Services v3, you use Azure Storage APIs to upload files. The following .NET snippet shows how.

The following function performs these actions:

- Creates an Asset
- Gets a writable [SAS URL](#) to the Asset's [container in storage](#)
- Uploads the file into the container in storage using the SAS URL

```

private static async Task<Asset> CreateInputAssetAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string assetName,
    string fileToUpload)
{
    // In this example, we are assuming that the asset name is unique.
    //
    // If you already have an asset with the desired name, use the Assets.Get method
    // to get the existing asset. In Media Services v3, the Get method on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).

    // Call Media Services API to create an Asset.
    // This method creates a container in storage for the Asset.
    // The files (blobs) associated with the asset will be stored in this container.
    Asset asset = await client.Assets.CreateOrUpdateAsync(resourceGroupName, accountName, assetName, new
Asset());

    // Use Media Services API to get back a response that contains
    // SAS URL for the Asset container into which to upload blobs.
    // That is where you would specify read-write permissions
    // and the expiration time for the SAS URL.
    var response = await client.Assets.ListContainerSasAsync(
        resourceGroupName,
        accountName,
        assetName,
        permissions: AssetContainerPermission.ReadWrite,
        expiryTime: DateTime.UtcNow.AddHours(4).ToUniversalTime());

    var sasUri = new Uri(response.AssetContainerSasUrls.First());

    // Use Storage API to get a reference to the Asset container
    // that was created by calling Asset's CreateOrUpdate method.
    CloudBlobContainer container = new CloudBlobContainer(sasUri);
    var blob = container.GetBlockBlobReference(Path.GetFileName(fileToUpload));

    // Use Storage API to upload the file into the container in storage.
    await blob.UploadFromFileAsync(fileToUpload);

    return asset;
}

```

Create an output asset to store the result of the job

The output [Asset](#) stores the result of your job. The project defines the **DownloadResults** function that downloads the results from this output asset into the "output" folder, so you can see what you got.

```
private static async Task<Asset> CreateOutputAssetAsync(IAzureMediaServicesClient client, string
resourceGroupName, string accountName, string assetName)
{
    // Check if an Asset already exists
    Asset outputAsset = await client.Assets.GetAsync(resourceGroupName, accountName, assetName);
    Asset asset = new Asset();
    string outputAssetName = assetName;

    if (outputAsset != null)
    {
        // Name collision! In order to get the sample to work, let's just go ahead and create a unique asset
name
        // Note that the returned Asset can have a different name than the one specified as an input
parameter.
        // You may want to update this part to throw an Exception instead, and handle name collisions
differently.
        string uniqueness = $"-{Guid.NewGuid().ToString("N")}";
        outputAssetName += uniqueness;

        Console.WriteLine("Warning - found an existing Asset with name = " + assetName);
        Console.WriteLine("Creating an Asset with this name instead: " + outputAssetName);
    }

    return await client.Assets.CreateOrUpdateAsync(resourceGroupName, accountName, outputAssetName, asset);
}
```

Create a transform and a job that analyzes videos

When encoding or processing content in Media Services, it is a common pattern to set up the encoding settings as a recipe. You would then submit a **Job** to apply that recipe to a video. By submitting new Jobs for each new video, you are applying that recipe to all the videos in your library. A recipe in Media Services is called as a **Transform**. For more information, see [Transforms and jobs](#). The sample described in this tutorial defines a recipe that analyzes the specified video.

Transform

When creating a new [Transform](#) instance, you need to specify what you want it to produce as an output. The required parameter is a **TransformOutput** object, as shown in the code above. Each **TransformOutput** contains a **Preset**. **Preset** describes step-by-step instructions of video and/or audio processing operations that are to be used to generate the desired **TransformOutput**. In this example, the **VideoAnalyzerPreset** preset is used and the language ("en-US") is passed to its constructor. This preset enables you to extract multiple audio and video insights from a video. You can use the **AudioAnalyzerPreset** preset if you need to extract multiple audio insights from a video.

When creating a **Transform**, you should first check if one already exists using the **Get** method, as shown in the code that follows. In Media Services v3, **Get** methods on entities return **null** if the entity doesn't exist (a case-insensitive check on the name).

```

private static async Task<Transform> GetOrCreateTransformAsync(IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    Preset preset)
{
    // Does a Transform already exist with the desired name? Assume that an existing Transform with the
    // desired name
    // also uses the same recipe or Preset for processing content.
    Transform transform = await client.Transforms.GetAsync(resourceGroupName, accountName, transformName);

    if (transform == null)
    {
        // Start by defining the desired outputs.
        TransformOutput[] outputs = new TransformOutput[]
        {
            new TransformOutput(preset),
        };

        // Create the Transform with the output defined above
        transform = await client.Transforms.CreateOrUpdateAsync(resourceGroupName, accountName, transformName,
            outputs);
    }

    return transform;
}

```

Job

As mentioned above, the [Transform](#) object is the recipe and a [Job](#) is the actual request to Media Services to apply that **Transform** to a given input video or audio content. The **Job** specifies information like the location of the input video, and the location for the output. You can specify the location of your video using: HTTPS URLs, SAS URLs, or Assets that are in your Media Service account.

In this example, the job input is a local video.

```

private static async Task<Job> SubmitJobAsync(IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string jobName,
    JobInput jobInput,
    string outputAssetName)
{
    JobOutput[] jobOutputs =
    {
        new JobOutputAsset(outputAssetName),
    };

    // In this example, we are assuming that the job name is unique.
    //
    // If you already have a job with the desired name, use the Jobs.Get method
    // to get the existing job. In Media Services v3, Get methods on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).
    Job job = await client.Jobs.CreateAsync(
        resourceGroupName,
        accountName,
        transformName,
        jobName,
        new Job
        {
            Input = jobInput,
            Outputs = jobOutputs,
        });

    return job;
}

```

Wait for the job to complete

The job takes some time to complete and when it does you want to be notified. There are different options to get notified about the [Job](#) completion. The simplest option (that is shown here) is to use polling.

Polling is not a recommended best practice for production applications because of potential latency. Polling can be throttled if overused on an account. Developers should instead use Event Grid.

Event Grid is designed for high availability, consistent performance, and dynamic scale. With Event Grid, your apps can listen for and react to events from virtually all Azure services, as well as custom sources. Simple, HTTP-based reactive event handling helps you build efficient solutions through intelligent filtering and routing of events. See [Route events to a custom web endpoint](#).

The **Job** usually goes through the following states: **Scheduled**, **Queued**, **Processing**, **Finished** (the final state). If the job has encountered an error, you get the **Error** state. If the job is in the process of being canceled, you get **Canceling** and **Canceled** when it is done.

```

private static async Task<Job> WaitForJobToFinishAsync(IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string jobName)
{
    const int SleepIntervalMs = 60 * 1000;

    Job job = null;

    do
    {
        job = await client.Jobs.GetAsync(resourceGroupName, accountName, transformName, jobName);

        Console.WriteLine($"Job is '{job.State}'");
        for (int i = 0; i < job.Outputs.Count; i++)
        {
            JobOutput output = job.Outputs[i];
            Console.WriteLine($"JobOutput[{i}] is '{output.State}'");
            if (output.State == JobState.Processing)
            {
                Console.WriteLine($"Progress: '{output.Progress}'");
            }

            Console.WriteLine();
        }

        if (job.State != JobState.Finished && job.State != JobState.Error && job.State != JobState.Canceled)
        {
            await Task.Delay(SleepIntervalMs);
        }
    }
    while (job.State != JobState.Finished && job.State != JobState.Error && job.State != JobState.Canceled);

    return job;
}

```

Download the result of the job

The following function downloads the results from the output [Asset](#) into the "output" folder, so you can examine the results of the job.


```

private static async Task DownloadOutputAssetAsync(
    IAzureMediaServicesClient client,
    string resourceGroup,
    string accountName,
    string assetName,
    string outputFolderName)
{
    const int ListBlobsSegmentMaxResult = 5;

    if (!Directory.Exists(outputFolderName))
    {
        Directory.CreateDirectory(outputFolderName);
    }

    AssetContainerSas assetContainerSas = await client.Assets.ListContainerSasAsync(
        resourceGroup,
        accountName,
        assetName,
        permissions: AssetContainerPermission.Read,
        expiryTime: DateTime.UtcNow.AddHours(1).ToUniversalTime());

    Uri containerSasUrl = new Uri(assetContainerSas.AssetContainerSasUrls.FirstOrDefault());
    CloudBlobContainer container = new CloudBlobContainer(containerSasUrl);

    string directory = Path.Combine(outputFolderName, assetName);
    Directory.CreateDirectory(directory);

    Console.WriteLine($"Downloading output results to '{directory}'...");

    BlobContinuationToken continuationToken = null;
    IList<Task> downloadTasks = new List<Task>();

    do
    {
        BlobResultSegment segment = await container.ListBlobsSegmentedAsync(null, true,
            BlobListingDetails.None, ListBlobsSegmentMaxResult, continuationToken, null, null);

        foreach (IListBlobItem blobItem in segment.Results)
        {
            CloudBlockBlob blob = blobItem as CloudBlockBlob;
            if (blob != null)
            {
                string path = Path.Combine(directory, blob.Name);

                downloadTasks.Add(blob.DownloadToFileAsync(path, FileMode.Create));
            }
        }

        continuationToken = segment.ContinuationToken;
    }
    while (continuationToken != null);

    await Task.WhenAll(downloadTasks);

    Console.WriteLine("Download complete.");
}

```

Clean up resource in your Media Services account

Generally, you should clean up everything except objects that you are planning to reuse (typically, you will reuse Transforms and persist StreamingLocators). If you want for your account to be clean after experimenting, you should delete the resources that you do not plan to reuse. For example, the following code deletes Jobs.

```
private static async Task CleanUpAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName)
{
    var jobs = await client.Jobs.ListAsync(resourceGroupName, accountName, transformName);
    foreach (var job in jobs)
    {
        await client.Jobs.DeleteAsync(resourceGroupName, accountName, transformName, job.Name);
    }

    var assets = await client.Assets.ListAsync(resourceGroupName, accountName);
    foreach (var asset in assets)
    {
        await client.Assets.DeleteAsync(resourceGroupName, accountName, asset.Name);
    }
}
```

Run the sample app

Press Ctrl+F5 to run the *AnalyzeVideos* application.

When we run the program, the job produces thumbnails for each face that it finds in the video. It also produces the insights.json file.

Examine the output

The output file of analyzing videos is called insights.json. This file contains insights about your video. You can find description of elements found in the json file in the [Media intelligence](#) article.

Clean up resources

If you no longer need any of the resources in your resource group, including the Media Services and storage accounts you created for this tutorial, delete the resource group you created earlier.

Execute the following CLI command:

```
az group delete --name amsResourceGroup
```

Multithreading

The Azure Media Services v3 SDKs are not thread-safe. When working with multi-threaded application, you should generate a new AzureMediaServicesClient object per thread.

Next steps

[Tutorial: upload, encode, and stream files](#)

Azure CLI examples for Azure Media Services

12/10/2018 • 2 minutes to read • [Edit Online](#)

The following table includes links to the Azure CLI examples for Azure Media Services.

Examples

Account	
Create a Media Services account	Creates an Azure Media Services account. Also, creates a service principal that can be used to access APIs to programmatically manage the account.
Reset account credentials	Resets your account credentials and gets the app.config settings back.
Assets	
Create assets	Creates a Media Services Asset to upload content to.
Upload a file	Uploads a local file to a storage container.
Publish an asset	Creates a Streaming Locator and gets Streaming URLs back.
Transforms and Jobs	
Create transforms	Shows how to create transforms. Transforms describe a simple workflow of tasks for processing your video or audio files (often referred to as a "recipe"). You should always check if a Transform with desired name and "recipe" already exist. If it does, reuse it.
Create jobs	Submits a Job to a simple encoding Transform using HTTPs URL.
Create EventGrid	Creates an account level Event Grid subscription for Job State Changes.

See also

[Azure CLI](#)

Migration guidance for moving from Media Services v2 to v3

12/19/2018 • 5 minutes to read • [Edit Online](#)

This article describes changes that were introduced in Azure Media Services v3, shows differences between two versions, and provides the migration guidance.

If you have a video service developed today on top of the [legacy Media Services v2 APIs](#), you should review the following guidelines and considerations prior to migrating to the v3 APIs. There are many benefits and new features in the v3 API that improve the developer experience and capabilities of Media Services. However, as called out in the [Known Issues](#) section of this article, there are also some limitations due to changes between the API versions. This page will be maintained as the Media Services team makes continued improvements to the v3 APIs and addresses the gaps between the versions.

NOTE

Currently, you cannot use the Azure portal to manage v3 resources. Use the [REST API](#), CLI, or one of the supported SDKs.

Benefits of Media Services v3

API is more approachable

- v3 is based on a unified API surface, which exposes both management and operations functionality built on Azure Resource Manager. Azure Resource Manager templates can be used to create and deploy Transforms, Streaming Endpoints, LiveEvents, and more.
- [Open API \(aka Swagger\) Specification](#) document. Exposes the schema for all service components, including file-based encoding.
- SDKs available for [.NET](#), [.NET Core](#), [Node.js](#), [Python](#), [Java](#), [Go](#), and [Ruby](#).
- [Azure CLI](#) integration for simple scripting support.

New features

- For file-based Job processing, you can use a HTTP(S) URL as the input. You do not need to have content already stored in Azure, nor do you need to create Assets.
- Introduces the concept of [Transforms](#) for file-based Job processing. A Transform can be used to build reusable configurations, to create Azure Resource Manager Templates, and isolate processing settings between multiple customers or tenants.
- An Asset can have [multiple StreamingLocators](#) each with different Dynamic Packaging and Dynamic Encryption settings.
- [Content protection](#) supports multi-key features.
- You can stream live events that are up to 24 hours long when using Media Services for transcoding a single bitrate contribution feed into an output stream that has multiple bitrates.
- New Low Latency live streaming support on LiveEvents.
- LiveEvent Preview supports Dynamic Packaging and Dynamic Encryption. This enables content protection on Preview as well as DASH and HLS packaging.
- LiveOutput is simpler to use than the Program entity in the v2 APIs.
- You have role-based access control (RBAC) over your entities.

Changes from v2

- For Assets created with v3, Media Services supports only the [Azure Storage server-side storage encryption](#).
 - You can use v3 APIs with Assets created with v2 APIs that had [storage encryption](#) (AES 256) provided by Media Services.
 - You cannot create new Assets with the legacy AES 256 [storage encryption](#) using v3 APIs.
- The v3 SDKs are now decoupled from the Storage SDK, which gives you more control over the version of Storage SDK you want to use and avoids versioning issues.
- In the v3 APIs, all of the encoding bit rates are in bits per second. This is different than the v2 Media Encoder Standard presets. For example, the bitrate in v2 would be specified as 128 (kbps), but in v3 it would be 128000 (bits/second).
- Entities AssetFiles, AccessPolicies, and IngestManifests do not exist in v3.
- The IAsset.ParentAssets property does not exist in v3.
- ContentKeys is no longer an entity, it is now a property of the StreamingLocator.
- Event Grid support replaces NotificationEndpoints.
- The following entities were renamed
 - JobOutput replaces Task, and is now part of a Job.
 - StreamingLocator replaces Locator.
 - LiveEvent replaces Channel.
LiveEvents billing is based on Live Channel meters. For more information, see [Live streaming overview](#) and [pricing](#).
 - LiveOutput replaces Program.
- LiveOutputs do not need to be started explicitly, they start on creation and stop when deleted. Programs worked differently in the v2 APIs, they had to be started after creation.

Feature gaps with respect to v2 APIs

The v3 API has the following feature gaps with respect to the v2 API. Closing the gaps is work in progress.

- The [Premium Encoder](#) and the legacy [media analytics processors](#) (Azure Media Services Indexer 2 Preview, Face Redactor, etc.) are not accessible via v3.
Customers who wish to migrate from the Media Indexer 1 or 2 preview can immediately use the AudioAnalyzer preset in the v3 API. This new preset contains more features than the older Media Indexer 1 or 2.
- Many of the advanced features of the Media Encoder Standard in v2 APIs are currently not available in v3, such as:
 - Clipping (for on-demand and live scenarios)
 - Stitching of Assets
 - Overlays
 - Cropping
 - Thumbnail Sprites
- LiveEvents with transcoding currently do not support Slate insertion mid-stream, custom presets, or ad marker insertion via API call.

NOTE

Please bookmark this article and keep checking for updates.

Code differences

The following table shows the code differences between v2 and v3 for common scenarios.

SCENARIO	V2 API	V3 API
Create an asset and upload a file	v2 .NET example	v3 .NET example
Submit a job	v2 .NET example	v3 .NET example Shows how to first create a Transform and then submit a Job.
Publish an asset with AES encryption	1. Create ContentKeyAuthorizationPolicyOption 2. Create ContentKeyAuthorizationPolicy 3. Create AssetDeliveryPolicy 4. Create Asset and upload content OR Submit job and use output asset 5. Associate AssetDeliveryPolicy with Asset 6. Create ContentKey 7. Attach ContentKey to Asset 8. Create AccessPolicy 9. Create Locator v2 .NET example	1. Create Content Key Policy 2. Create Asset 3. Upload content or use Asset as JobOutput 4. Create StreamingLocator v3 .NET example

Known issues

- Currently, you cannot use the Azure portal to manage v3 resources. Use the [REST API](#), CLI, or one of the supported SDKs.
- You need to provision Media Reserved Units (MRUs) in your account in order to control the concurrency and performance of your Jobs, particularly ones involving Video or Audio Analysis. For more information, see [Scaling Media Processing](#). You can manage the MRUs using [CLI 2.0 for Media Services v3](#), using the [Azure portal](#), or using the [v2 APIs](#). You need to provision MRUs, whether you are using Media Services v2 or v3 APIs.
- Media Services entities created with the v3 API cannot be managed by the v2 API.
- It is not recommended to manage entities that were created with v2 APIs via the v3 APIs. Following are examples of the differences that make the entities in two versions incompatible:
 - Jobs and Tasks created in v2 do not show up in v3 as they are not associated with a Transform. The recommendation is to switch to v3 Transforms and Jobs. There will be a relatively short time period of needing to monitor the inflight v2 Jobs during the switchover.
 - Channels and Programs created with v2 (which are mapped to LiveEvents and LiveOutputs in v3) cannot continue being managed with v3. The recommendation is to switch to v3 LiveEvents and LiveOutputs at a convenient Channel Stop. Presently, you cannot migrate continuously running Channels.

NOTE

This page will be maintained as the Media Services team makes continued improvements to the v3 APIs and addresses the gaps between the versions.

Next steps

To see how easy it is to start encoding and streaming video files, check out [Stream files](#).

Analyzing video and audio files

12/13/2018 • 8 minutes to read • [Edit Online](#)

Azure Media Services v3 enables you to extract insights from your video and audio files with Video Indexer through AMS v3 analyzer presets (described in this article). If you want more detailed insights, use Video Indexer directly. To understand when you would want to use Video Indexer vs. Media Services analyzer presets, check out the [comparison document](#).

To analyze your content using Media Services v3 presets, you create a **Transform** and submit a **Job** that uses one of these presets: **AudioAnalyzerPreset** or **VideoAnalyzerPreset**. The following article demonstrates how to use **VideoAnalyzerPreset**: [Tutorial: Analyze videos with Azure Media Services](#).

NOTE

When using a Video or Audio Analyzer presets, use the Azure portal to set your account to have 10 S3 Media Reserved Units. For more information, see [Scale media processing](#).

Built-in presets

Media Services currently supports the following built-in analyzer presets:

PRESET NAME	SCENARIO	DETAILS
AudioAnalyzerPreset	Analyzing audio	<p>The preset applies a pre-defined set of AI-based analysis operations, including speech transcription. Currently, the preset supports processing of content with a single audio track. You can specify the language for the audio payload in the input using the BCP-47 format of 'language tag-region'. Supported languages are English ('en-US' and 'en-GB'), Spanish ('es-ES' and 'es-MX'), French ('fr-FR'), Italian ('it-IT'), Japanese ('ja-JP'), Portuguese ('pt-BR'), Chinese ('zh-CN'), German ('de-DE'), Arabic ('ar-EG'), Russian ('ru-RU'), Hindi ('hi-IN'), and Korean ('ko-KR').</p> <p>If the language isn't specified or set to null, automatic language detection will be employed. The automatic language detection feature currently supports English, Chinese, French, German, Italian, Japanese, Spanish, Russian, and Portuguese. The automatic language detection feature works best with audio recordings with clearly discernable speech. If automatic language detection fails to find the language, the transcription will fall back to English.</p>

PRESET NAME	SCENARIO	DETAILS
VideoAnalyzerPreset	Analyzing audio and video	Extracts insights (rich metadata) from both audio and video, and outputs a JSON format file. You can specify whether you only want to extract audio insights when processing a video file. For more information, see Analyze video .

AudioAnalyzerPreset

The preset enables you to extract multiple audio insights from an audio or video file. The output includes a JSON file (with all the insights) and VTT file for the audio transcript. This preset accepts a property that specifies the language of the input file in the form of a [BCP47](#) string. The audio insights include:

- Audio transcription – a transcript of the spoken words with timestamps. Multiple languages are supported
- Speaker indexing – a mapping of the speakers and the corresponding spoken words
- Speech sentiment analysis – the output of sentiment analysis performed on the audio transcription
- Keywords – keywords that are extracted from the audio transcription.

VideoAnalyzerPreset

The preset enables you to extract multiple audio and video insights from a video file. The output includes a JSON file (with all the insights), a VTT file for the video transcript, and a collection of thumbnails. This preset also accepts a [BCP47](#) string (representing the language of the video) as a property. The video insights include all the audio insights mentioned above and the following additional items:

- Face tracking – the time during which faces are present in the video. Each face has a face id and a corresponding collection of thumbnails
- Visual text – the text that is detected via optical character recognition. The text is time stamped and also used to extract keywords (in addition to the audio transcript)
- Keyframes – a collection of keyframes that are extracted from the video
- Visual content moderation – the portion of the videos that have been flagged as adult or racy in nature
- Annotation – a result of annotating the videos based on a pre-defined object model

insights.json elements

The output includes a JSON file (insights.json) with all the insights that were found in the video or audio. The json may contain the following elements:

transcript

NAME	DESCRIPTION
id	The line ID.
text	The transcript itself.
language	The transcript language. Intended to support transcript where each line can have a different language.
instances	A list of time ranges where this line appeared. If the instance is transcript, it will have only 1 instance.

Example:


```
"transcript": [
{
  "id": 0,
  "text": "Hi I'm Doug from office.",
  "language": "en-US",
  "instances": [
    {
      "start": "00:00:00.510000",
      "end": "00:00:02.720000"
    }
  ]
},
{
  "id": 1,
  "text": "I have a guest. It's Michelle.",
  "language": "en-US",
  "instances": [
    {
      "start": "00:00:02.720000",
      "end": "00:00:03.960000"
    }
  ]
}
]
```

OCR

NAME	DESCRIPTION
id	The OCR line ID.
text	The OCR text.
confidence	The recognition confidence.
language	The OCR language.
instances	A list of time ranges where this OCR appeared (the same OCR can appear multiple times).

```

"ocr": [
  {
    "id": 0,
    "text": "LIVE FROM NEW YORK",
    "confidence": 0.91,
    "language": "en-US",
    "instances": [
      {
        "start": "00:00:26",
        "end": "00:00:52"
      }
    ]
  },
  {
    "id": 1,
    "text": "NOTICIAS EN VIVO",
    "confidence": 0.9,
    "language": "es-ES",
    "instances": [
      {
        "start": "00:00:26",
        "end": "00:00:28"
      },
      {
        "start": "00:00:32",
        "end": "00:00:38"
      }
    ]
  }
],

```

faces

NAME	DESCRIPTION
id	The face ID.
name	The face name. It can be 'Unknown #0', an identified celebrity or a customer trained person.
confidence	The face identification confidence.
description	A description of the celebrity.
thumbnailId	The ID of the thumbnail of that face.
knownPersonId	If it is a known person, its internal ID.
referenceId	If it is a Bing celebrity, its Bing ID.
referenceType	Currently just Bing.
title	If it is a celebrity, its title (for example "Microsoft's CEO").
imageUrl	If it is a celebrity, its image url.
instances	These are instances of where the face appeared in the given time range. Each instance also has a thumbnailId.

```
"faces": [{
  "id": 2002,
  "name": "Xam 007",
  "confidence": 0.93844,
  "description": null,
  "thumbnailId": "00000000-ae4-4be2-a4d5-d01817c07955",
  "knownPersonId": "8340004b-5cf5-4611-9cc4-3b13cca10634",
  "referenceId": null,
  "title": null,
  "imageUrl": null,
  "instances": [{
    "thumbnailsIds": ["00000000-9f68-4bb2-ab27-3b4d9f2d998e",
      "cef03f24-b0c7-4145-94d4-a84f81bb588c"],
    "adjustedStart": "00:00:07.240000",
    "adjustedEnd": "00:00:45.678000",
    "start": "00:00:07.240000",
    "end": "00:00:45.678000"
  }],
  {
    "thumbnailsIds": ["00000000-51e5-4260-91a5-890fa05c68b0"],
    "adjustedStart": "00:10:23.957000",
    "adjustedEnd": "00:10:39.239000",
    "start": "00:10:23.957000",
    "end": "00:10:39.239000"
  }
}]
}]
```

shots

NAME	DESCRIPTION
id	The shot ID.
keyFrames	A list of key frames within the shot (each has an ID and a list of instances time ranges). Key frames instances have a thumbnailId field with the keyFrame's thumbnail ID.
instances	A list of time ranges of this shot (shots have only 1 instance).

```

"Shots": [
  {
    "id": 0,
    "keyFrames": [
      {
        "id": 0,
        "instances": [
          {
            "thumbnailId": "00000000-0000-0000-0000-000000000000",
            "start": "00: 00: 00.1670000",
            "end": "00: 00: 00.2000000"
          }
        ]
      }
    ],
    "instances": [
      {
        "thumbnailId": "00000000-0000-0000-0000-000000000000",
        "start": "00: 00: 00.2000000",
        "end": "00: 00: 05.0330000"
      }
    ]
  },
  {
    "id": 1,
    "keyFrames": [
      {
        "id": 1,
        "instances": [
          {
            "thumbnailId": "00000000-0000-0000-0000-000000000000",
            "start": "00: 00: 05.2670000",
            "end": "00: 00: 05.3000000"
          }
        ]
      }
    ],
    "instances": [
      {
        "thumbnailId": "00000000-0000-0000-0000-000000000000",
        "start": "00: 00: 05.2670000",
        "end": "00: 00: 10.3000000"
      }
    ]
  }
]

```

statistics

NAME	DESCRIPTION
CorrespondenceCount	Number of correspondences in the video.
WordCount	The number of words per speaker.
SpeakerNumberOfFragments	The amount of fragments the speaker has in a video.
SpeakerLongestMonolog	The speaker's longest monolog. If the speaker has silences inside the monolog it is included. Silence at the beginning and the end of the monolog is removed.

NAME	DESCRIPTION
SpeakerTalkToListenRatio	The calculation is based on the time spent on the speaker's monolog (without the silence in between) divided by the total time of the video. The time is rounded to the third decimal point.

sentiments

Sentiments are aggregated by their sentimentType field (Positive/Neutral/Negative). For example, 0-0.1, 0.1-0.2.

NAME	DESCRIPTION
id	The sentiment ID.
averageScore	The average of all scores of all instances of that sentiment type - Positive/Neutral/Negative
instances	A list of time ranges where this sentiment appeared.
sentimentType	The type can be 'Positive', 'Neutral', or 'Negative'.

```
"sentiments": [
{
  "id": 0,
  "averageScore": 0.87,
  "sentimentType": "Positive",
  "instances": [
    {
      "start": "00:00:23",
      "end": "00:00:41"
    }
  ]
}, {
  "id": 1,
  "averageScore": 0.11,
  "sentimentType": "Positive",
  "instances": [
    {
      "start": "00:00:13",
      "end": "00:00:21"
    }
  ]
}
]
```

labels

NAME	DESCRIPTION
id	The label ID.
name	The label name (for example, 'Computer', 'TV').
language	The label name language (when translated). BCP-47
instances	A list of time ranges where this label appeared (a label can appear multiple times). Each instance has a confidence field.

```

"labels": [
  {
    "id": 0,
    "name": "person",
    "language": "en-US",
    "instances": [
      {
        "confidence": 1.0,
        "start": "00: 00: 00.0000000",
        "end": "00: 00: 25.6000000"
      },
      {
        "confidence": 1.0,
        "start": "00: 01: 33.8670000",
        "end": "00: 01: 39.2000000"
      }
    ]
  },
  {
    "name": "indoor",
    "language": "en-US",
    "id": 1,
    "instances": [
      {
        "confidence": 1.0,
        "start": "00: 00: 06.4000000",
        "end": "00: 00: 07.4670000"
      },
      {
        "confidence": 1.0,
        "start": "00: 00: 09.6000000",
        "end": "00: 00: 10.6670000"
      },
      {
        "confidence": 1.0,
        "start": "00: 00: 11.7330000",
        "end": "00: 00: 20.2670000"
      },
      {
        "confidence": 1.0,
        "start": "00: 00: 21.3330000",
        "end": "00: 00: 25.6000000"
      }
    ]
  }
]

```

keywords

NAME	DESCRIPTION
id	The keyword ID.
text	The keyword text.
confidence	The keyword's recognition confidence.
language	The keyword language (when translated).
instances	A list of time ranges where this keyword appeared (a keyword can appear multiple times).

```

"keywords": [
{
  "id": 0,
  "text": "office",
  "confidence": 1.6666666666666667,
  "language": "en-US",
  "instances": [
    {
      "start": "00:00:00.510000",
      "end": "00:00:02.720000"
    },
    {
      "start": "00:00:03.960000",
      "end": "00:00:12.270000"
    }
  ]
},
{
  "id": 1,
  "text": "icons",
  "confidence": 1.4,
  "language": "en-US",
  "instances": [
    {
      "start": "00:00:03.960000",
      "end": "00:00:12.270000"
    },
    {
      "start": "00:00:13.990000",
      "end": "00:00:15.610000"
    }
  ]
}
]

```

visualContentModeration

The visualContentModeration block contains time ranges which Video Indexer found to potentially have adult content. If visualContentModeration is empty, there is no adult content that was identified.

Videos that are found to contain adult or racy content might be available for private view only. Users have the option to submit a request for a human review of the content, in which case the IsAdult attribute will contain the result of the human review.

NAME	DESCRIPTION
id	The visual content moderation ID.
adultScore	The adult score (from content moderator).
racyScore	The racy score (from content moderation).
instances	A list of time ranges where this visual content moderation appeared.

```
"VisualContentModeration": [  
  {  
    "id": 0,  
    "adultScore": 0.00069,  
    "racyScore": 0.91129,  
    "instances": [  
      {  
        "start": "00:00:25.4840000",  
        "end": "00:00:25.5260000"  
      }  
    ]  
  },  
  {  
    "id": 1,  
    "adultScore": 0.99231,  
    "racyScore": 0.99912,  
    "instances": [  
      {  
        "start": "00:00:35.5360000",  
        "end": "00:00:35.5780000"  
      }  
    ]  
  }  
]
```

Next steps

[Tutorial: Analyze videos with Azure Media Services](#)

Encoding with Media Services

12/10/2018 • 2 minutes to read • [Edit Online](#)

Azure Media Services enables you to encode your high-quality digital media files into formats that can be played on a wide variety of browsers and devices. For example, you might want to stream your content in Apple's HLS or MPEG DASH formats. This topic gives you guidance on how to encode your content with Media Services v3.

To encode with Media Services v3, you need to create a transform and a job. A transform defines the recipe for your encoding settings and outputs, and the job is an instance of the recipe. For more information, see [Transforms and Jobs](#)

When encoding with Media Services, you use presets to tell the encoder how the input media files should be processed. For example, you can specify the video resolution and/or the number of audio channels you want in the encoded content.

You can get started quickly with one of the recommended built-in presets based on industry best practices or you can choose to build a custom preset to target your specific scenario or device requirements. For more information, see [Encode with a custom Transform](#).

Built-in presets

Media Services currently supports the following built-in encoding presets:

PRESET NAME	SCENARIO	DETAILS
-------------	----------	---------

PRESET NAME	SCENARIO	DETAILS
BuiltInStandardEncoderPreset	Streaming	<p>Used to set a built-in preset for encoding the input video with the Standard Encoder.</p> <p>The following presets are currently supported:</p> <p>EncoderNamedPreset.AdaptiveStreaming (recommended). For more information, see auto-generating a bitrate ladder.</p> <p>EncoderNamedPreset.AACGoodQualityAudio - produces a single MP4 file containing only stereo audio encoded at 192 kbps.</p> <p>EncoderNamedPreset.H264MultipleBitrate1080p - produces a set of 8 GOP-aligned MP4 files, ranging from 6000 kbps to 400 kbps, and stereo AAC audio. Resolution starts at 1080p and goes down to 360p.</p> <p>EncoderNamedPreset.H264MultipleBitrate720p - produces a set of 6 GOP-aligned MP4 files, ranging from 3400 kbps to 400 kbps, and stereo AAC audio. Resolution starts at 720p and goes down to 360p.</p> <p>EncoderNamedPreset.H264MultipleBitrateSD - produces a set of 5 GOP-aligned MP4 files, ranging from 1600kbps to 400 kbps, and stereo AAC audio. Resolution starts at 480p and goes down to 360p.</p> <p>For more information, see Uploading, encoding, and streaming files.</p>
StandardEncoderPreset	Streaming	<p>Describes settings to be used when encoding the input video with the Standard Encoder.</p> <p>Use this preset when customizing Transform presets. For more information, see How to customize Transform presets.</p>

Custom presets

Media Services fully supports customizing all values in presets to meet your specific encoding needs and requirements. You use the **StandardEncoderPreset** preset when customizing Transform presets. For a detailed explanations and example, see [How to customize encoder presets](#).

Scaling encoding in v3

Currently, customers have to use the Azure portal or Media Services v2 APIs to set RUs (as described in [Scaling media processing](#)).

Next steps

Tutorials

The following tutorial shows how to encode your content with Media Services:

- [Upload, encode, and stream using Media Services](#)

Code samples

The following code samples contain code that shows how to encode with Media Services:

- [.NET Core](#)
- [Azure CLI](#)

SDKs

You can use any of the following supported Media Services v3 SDKs to encode your content.

- [Azure CLI](#)
- [REST](#)
- [.NET](#)
- [Java](#)
- [Python](#)

Media Encoder Standard formats and codecs

12/13/2018 • 2 minutes to read • [Edit Online](#)

This article contains a list of the most common import and export file formats that you can use with [StandardEncoderPreset](#). For information on how to create custom presets using **StandardEncoderPreset**, see [Create a transform with a custom preset](#).

Input container/file formats

FILE FORMATS (FILE EXTENSIONS)	SUPPORTED
FLV (with H.264 and AAC codecs) (.flv)	Yes
MXF (.mxf)	Yes
GXF (.gxf)	Yes
MPEG2-PS, MPEG2-TS, 3GP (.ts, .ps, .3gp, .3gpp, .mpg)	Yes
Windows Media Video (WMV)/ASF (.wmv, .asf)	Yes
AVI (Uncompressed 8bit/10bit) (.avi)	Yes
MP4 (.mp4, .m4a, .m4v)/ISMV (.isma, .ismv)	Yes
Microsoft Digital Video Recording(DVR-MS) (.dvr-ms)	Yes
Matroska/WebM (.mkv)	Yes
WAVE/WAV (.wav)	Yes
QuickTime (.mov)	Yes

NOTE

Above is a list of the more commonly encountered file extensions. Media Encoder Standard does support many others (for example: .m2ts, .mpeg2video, .qt). If you try to encode a file and you get an error message about the format not being supported, provide your feedback [here](#).

Audio formats in input containers

Media Encoder Standard supports carrying the following audio formats in input containers:

- MXF, GXF, and QuickTime files, which have audio tracks with interleaved stereo or 5.1 samples

or

- MXF, GXF, and QuickTime files where the audio is carried as separate PCM tracks but the channel mapping (to stereo or 5.1) can be deduced from the file metadata

Input video codecs

INPUT VIDEO CODECS	SUPPORTED
AVC 8-bit/10-bit, up to 4:2:2, including AVCIntra	8 bit 4:2:0 and 4:2:2
Avid DNxHD (in MXF)	Yes
DVCPro/DVCProHD (in MXF)	Yes
Digital video (DV) (in AVI files)	Yes
JPEG 2000	Yes
MPEG-2 (up to 422 Profile and High Level; including variants such as XDCAM, XDCAM HD, XDCAM IMX, CableLabs®, and D10)	Up to 422 Profile
MPEG-1	Yes
VC-1/WMV9	Yes
Canopus HQ/HQX	No
MPEG-4 Part 2	Yes
Theora	Yes
YUV420 uncompressed, or mezzanine	Yes
Apple ProRes 422	Yes
Apple ProRes 422 LT	Yes
Apple ProRes 422 HQ	Yes
Apple ProRes Proxy	Yes
Apple ProRes 4444	Yes
Apple ProRes 4444 XQ	Yes
HEVC/H.265	Main Profile

Input audio codecs

INPUT AUDIO CODECS	SUPPORTED
AAC (AAC-LC, AAC-HE, and AAC-HEv2; up to 5.1)	Yes
MPEG Layer 2	Yes

INPUT AUDIO CODECS	SUPPORTED
MP3 (MPEG-1 Audio Layer 3)	Yes
Windows Media Audio	Yes
WAV/PCM	Yes
FLAC	Yes
Opus	Yes
Vorbis	Yes
AMR (adaptive multi-rate)	Yes
AES (SMPTE 331M and 302M, AES3-2003)	No
Dolby® E	No
Dolby® Digital (AC3)	No
Dolby® Digital Plus (E-AC3)	No

Output formats and codecs

The following table lists the codecs and file formats that are supported for export.

FILE FORMAT	VIDEO CODEC	AUDIO CODEC
MP4 (including multi-bitrate MP4 containers)	H.264 (High, Main, and Baseline Profiles)	AAC-LC, HE-AAC v1, HE-AAC v2
MPEG2-TS	H.264 (High, Main, and Baseline Profiles)	AAC-LC, HE-AAC v1, HE-AAC v2

Next steps

[Create a transform with a custom preset](#)

Encode with an auto-generated bitrate ladder

12/10/2018 • 2 minutes to read • [Edit Online](#)

Overview

This article explains how to use the Standard Encoder in Media Services to encode an input video into an auto-generated bitrate ladder (bitrate-resolution pairs) based on the input resolution and bitrate. This built-in encoder setting, or preset, will never exceed the input resolution and bitrate. For example, if the input is 720p at 3 Mbps, output remains 720p at best, and will start at rates lower than 3 Mbps.

Encoding for streaming

When you use the **AdaptiveStreaming** preset in **Transform**, you get an output that is suitable for delivery via streaming protocols like HLS and DASH. When using this preset, the service intelligently determines how many video layers to generate and at what bitrate and resolution. The output content contains MP4 files where AAC-encoded audio and H.264-encoded video is not interleaved.

To see an example of how this preset is used, see [Stream a file](#).

Output

This section shows three examples of the output video layers produced by the Media Services encoder as a result of encoding with the **AdaptiveStreaming** preset. In all cases, the output contains an audio-only MP4 file with stereo audio encoded at 128 kbps.

Example 1

Source with height "1080" and framerate "29.970" produces 6 video layers:

LAYER	HEIGHT	WIDTH	BITRATE (KBPS)
1	1080	1920	6780
2	720	1280	3520
3	540	960	2210
4	360	640	1150
5	270	480	720
6	180	320	380

Example 2

Source with height "720" and framerate "23.970" produces 5 video layers:

LAYER	HEIGHT	WIDTH	BITRATE (KBPS)
1	720	1280	2940
2	540	960	1850

LAYER	HEIGHT	WIDTH	BITRATE (KBPS)
3	360	640	960
4	270	480	600
5	180	320	320

Example 3

Source with height "360" and framerate "29.970" produces 3 video layers:

LAYER	HEIGHT	WIDTH	BITRATE (KBPS)
1	360	640	700
2	270	480	440
3	180	320	230

Next steps

[Stream a file](#)

Filters and dynamic manifests

11/29/2018 • 13 minutes to read • [Edit Online](#)

When delivering your content to customers (streaming Live events or Video on Demand) your client might need more flexibility than what's described in the default asset's manifest file. Azure Media Services enables you to define account filters and asset filters for your content.

Filters are server-side rules that allow your customers to do things like:

- Play back only a section of a video (instead of playing the whole video). For example:
 - Reduce the manifest to show a subclip of a live event ("sub-clip filtering"), or
 - Trim the start of a video ("trimming a video").
- Deliver only the specified renditions and/or specified language tracks that are supported by the device that is used to play back the content ("rendition filtering").
- Adjust Presentation Window (DVR) in order to provide a limited length of the DVR window in the player ("adjusting presentation window").

This topic describes [Concepts](#) and [shows filters definitions](#). It then gives details about [common scenarios](#). At the end of the article, you find links that show how to create filters programmatically.

Concepts

Dynamic manifests

Media Services offers **Dynamic Manifests** based on pre-defined [filters](#). Once you define filters, your clients could use them to stream a specific rendition or sub-clips of your video. They would specify filter(s) in the streaming URL. Filters could be applied to adaptive bitrate streaming protocols: Apple HTTP Live Streaming (HLS), MPEG-DASH, and Smooth Streaming.

The following table shows some examples of URLs with filters:

PROTOCOL	EXAMPLE
HLS V4	<code>http://testendpoint-testaccount.streaming.mediaservices.windows.net/fecebb23-46f6-490d-8b70-203e86b0df58/BigBuckBunny.ism/Manifest(format=m3u8-aapl, filter=myAccountFilter)</code>
HLS V3	<code>http://testendpoint-testaccount.streaming.mediaservices.windows.net/fecebb23-46f6-490d-8b70-203e86b0df58/BigBuckBunny.ism/Manifest(format=m3u8-aapl-v3, filter=myAccountFilter)</code>
MPEG DASH	<code>http://testendpoint-testaccount.streaming.mediaservices.windows.net/fecebb23-46f6-490d-8b70-203e86b0df58/BigBuckBunny.ism/Manifest(format=mpd-time-csf, filter=myAssetFilter)</code>
Smooth Streaming	<code>http://testendpoint-testaccount.streaming.mediaservices.windows.net/fecebb23-46f6-490d-8b70-203e86b0df58/BigBuckBunny.ism/Manifest(filter=myAssetFilter)</code>

NOTE

Dynamic Manifests do not change the asset and the default manifest for that asset. Your client can choose to request a stream with or without filters.

Manifest files

When you encode an asset for adaptive bitrate streaming, a **manifest** (playlist) file is created (the file is text-based or XML-based). The **manifest** file includes streaming metadata such as: track type (audio, video, or text), track name, start and end time, bitrate (qualities), track languages, presentation window (sliding window of fixed duration), video codec (FourCC). It also instructs the player to retrieve the next fragment by providing information about the next playable video fragments available and their location. Fragments (or segments) are the actual "chunks" of a video content.

Here is an example of an HLS manifest file:

```
#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="audio",NAME="aac_eng_2_128041_2_1",LANGUAGE="eng",DEFAULT=YES,AUTOSELECT=YES,URI="QualityLevels(128041)/Manifest(aac_eng_2_128041_2_1,format=m3u8-aapl)"
#EXT-X-STREAM-INF:BANDWIDTH=536209,RESOLUTION=320x180,CODECS="avc1.64000d,mp4a.40.2",AUDIO="audio"
QualityLevels(380658)/Manifest(video,format=m3u8-aapl)
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=536209,RESOLUTION=320x180,CODECS="avc1.64000d",URI="QualityLevels(380658)/Manifest(video,format=m3u8-aapl,type=keyframes)"
#EXT-X-STREAM-INF:BANDWIDTH=884474,RESOLUTION=480x270,CODECS="avc1.640015,mp4a.40.2",AUDIO="audio"
QualityLevels(721426)/Manifest(video,format=m3u8-aapl)
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=884474,RESOLUTION=480x270,CODECS="avc1.640015",URI="QualityLevels(721426)/Manifest(video,format=m3u8-aapl,type=keyframes)"
#EXT-X-STREAM-INF:BANDWIDTH=1327838,RESOLUTION=640x360,CODECS="avc1.64001e,mp4a.40.2",AUDIO="audio"
QualityLevels(1155246)/Manifest(video,format=m3u8-aapl)
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=1327838,RESOLUTION=640x360,CODECS="avc1.64001e",URI="QualityLevels(1155246)/Manifest(video,format=m3u8-aapl,type=keyframes)"
#EXT-X-STREAM-INF:BANDWIDTH=2414544,RESOLUTION=960x540,CODECS="avc1.64001f,mp4a.40.2",AUDIO="audio"
QualityLevels(2218559)/Manifest(video,format=m3u8-aapl)
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=2414544,RESOLUTION=960x540,CODECS="avc1.64001f",URI="QualityLevels(2218559)/Manifest(video,format=m3u8-aapl,type=keyframes)"
#EXT-X-STREAM-INF:BANDWIDTH=3805301,RESOLUTION=1280x720,CODECS="avc1.640020,mp4a.40.2",AUDIO="audio"
QualityLevels(3579378)/Manifest(video,format=m3u8-aapl)
#EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=3805301,RESOLUTION=1280x720,CODECS="avc1.640020",URI="QualityLevels(3579378)/Manifest(video,format=m3u8-aapl,type=keyframes)"
#EXT-X-STREAM-INF:BANDWIDTH=139017,CODECS="mp4a.40.2",AUDIO="audio"
QualityLevels(128041)/Manifest(aac_eng_2_128041_2_1,format=m3u8-aapl)
```

Get and examine manifest files

You specify a list of filter track property conditions based on which the tracks of your stream (Live or Video on Demand) should be included into dynamically created manifest. To get and examine the properties of the tracks, you have to load the Smooth Streaming manifest first.

The [Upload, encode, and stream files with .NET](#) tutorial shows you how to build the streaming URLs with .NET (see, the [building URLs](#) section). If you run the app, one of the URLs points to the Smooth Streaming manifest:

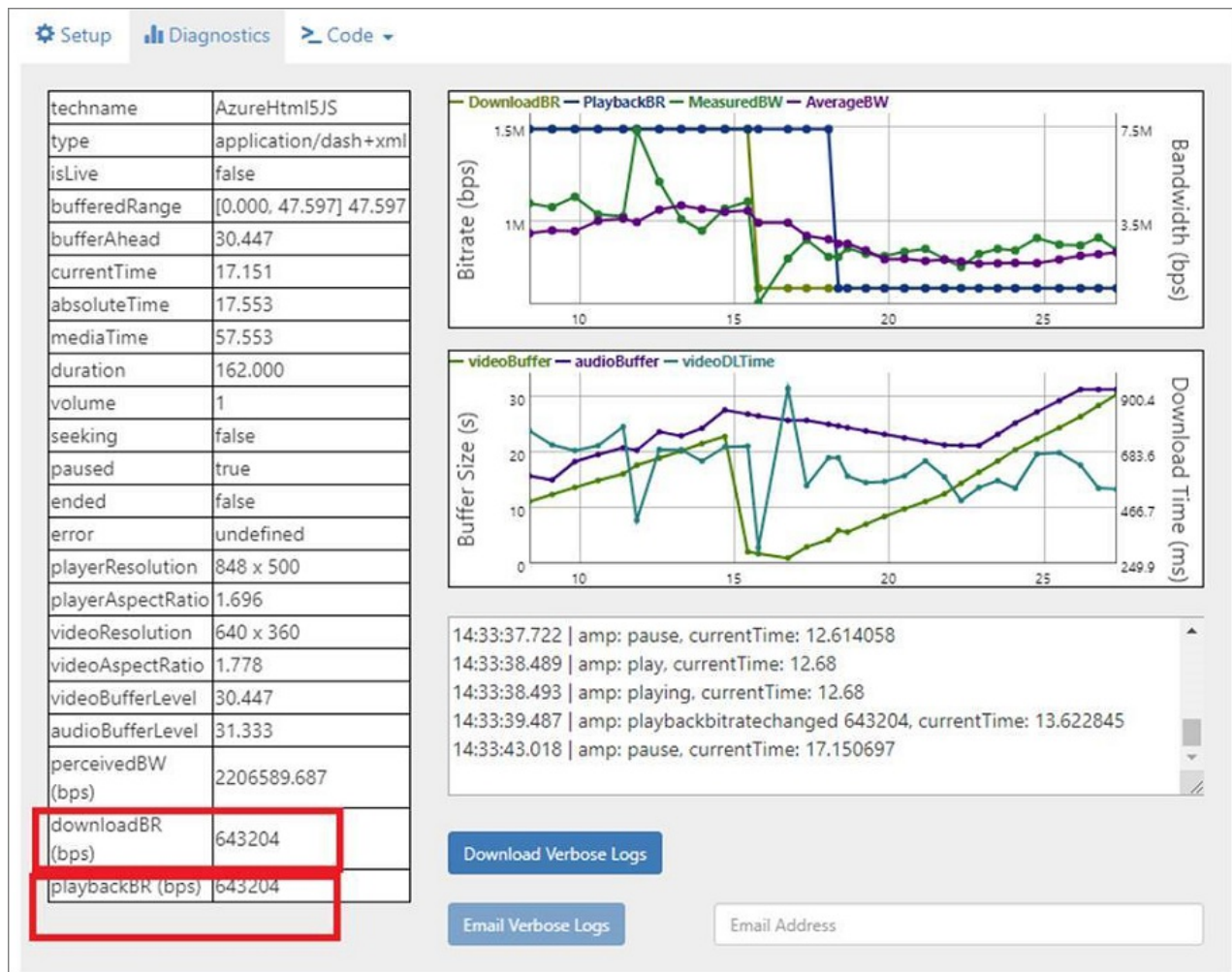
```
https://amsaccount-usw22.streaming.media.azure.net/00000000-0000-0000-0000-000000000000/ignite.ism/manifest.
```

Copy and paste the URL into the address bar of a browser. The file will be downloaded. You can open it in a text editor of your choice.

For the REST example, see [Upload, encode, and stream files with REST](#).

Monitor the bitrate of a video stream

You can use the [Azure Media Player demo page](#) to monitor the bitrate of a video stream. The demo page displays diagnostics info in the **Diagnostics** tab:



Defining filters

There are two types of asset filters:

- [Account Filters](#) (global) - can be applied to any asset in the Azure Media Services account, have a lifetime of the account.
- [Asset Filters](#) (local) - can only be applied to an asset with which the filter was associated upon creation, have a lifetime of the asset.

[Account Filter](#) and [Asset Filter](#) types have exactly the same properties for defining/describing the filter. Except when creating the **Asset Filter**, you need to specify the asset name with which you want to associate the filter.

Depending on your scenario, you decide what type of a filter is more suitable (Asset Filter or Account Filter). Account Filters are suitable for device profiles (rendition filtering) where Asset Filters could be used to trim a specific asset.

You use the following properties to describe the filters.

NAME	DESCRIPTION
firstQuality	The first quality bitrate of the filter.

NAME	DESCRIPTION
presentationTimeRange	The presentation time range. This property is used for filtering manifest start/end points, presentation window length, and the live start position. For more information, see PresentationTimeRange .
tracks	The tracks selection conditions. For more information, see tracks

PresentationTimeRange

Use this property with **Asset Filters**. It is not recommended to set the property with **Account Filters**.

NAME	DESCRIPTION
endTimeStamp	<p>The absolute end time boundary. Applies to Video on Demand (VoD). For the Live presentation, it is silently ignored and applied when the presentation ends and the stream becomes VoD.</p> <p>The value represents an absolute end point of the stream. It gets rounded to the closest next GOP start.</p> <p>Use StartTimestamp and EndTimestamp to trim the playlist (manifest). For example, StartTimestamp=40000000 and EndTimestamp = 100000000 will generate a playlist that contains media between StartTimestamp and EndTimestamp. If a fragment straddles the boundary, the entire fragment will be included in the manifest.</p> <p>Also, see the forceEndTimestamp definition that follows.</p>
forceEndTimestamp	<p>Applies to Live filters.</p> <p>forceEndTimestamp is a boolean that indicates whether or not endTimeStamp was set to a valid value.</p> <p>If the value is true, the endTimeStamp value should be specified. If it is not specified, then a bad request is returned.</p> <p>If for example, you want to define a filter that starts at 5 minutes into the input video, and lasts until the end of the stream, you would set forceEndTimestamp to false and omit setting endTimeStamp.</p>
liveBackoffDuration	<p>Applies to Live only. The property is used to define live playback position. Using this rule, you can delay live playback position and create a server-side buffer for players. LiveBackoffDuration is relative to the live position. The maximum live backoff duration is 60 seconds.</p>
presentationWindowDuration	<p>Applies to Live. Use presentationWindowDuration to apply a sliding window to the playlist. For example, set presentationWindowDuration=1200000000 to apply a two-minute sliding window. Media within 2 minutes of the live edge will be included in the playlist. If a fragment straddles the boundary, the entire fragment will be included in the playlist. The minimum presentation window duration is 120 seconds.</p>

NAME	DESCRIPTION
startTimestamp	<p>Applies to VoD or Live streams. The value represents an absolute start point of the stream. The value gets rounded to the closest next GOP start.</p> <p>Use startTimestamp and endTimestamp to trim the playlist (manifest). For example, startTimestamp=40000000 and endTimestamp = 100000000 will generate a playlist that contains media between StartTimestamp and EndTimestamp. If a fragment straddles the boundary, the entire fragment will be included in the manifest.</p>
timescale	<p>Applies to VoD or Live streams. The timescale used by the timestamps and durations specified above. The default timescale is 10000000. An alternative timescale can be used. Default is 10000000 HNS (hundred nanosecond).</p>

Tracks

You specify a list of filter track property conditions (FilterTrackPropertyConditions) based on which the tracks of your stream (Live or Video on Demand) should be included into dynamically created manifest. The filters are combined using a logical **AND** and **OR** operation.

Filter track property conditions describe track types, values (described in the following table), and operations (Equal, NotEqual).

NAME	DESCRIPTION
Bitrate	<p>Use the bitrate of the track for filtering.</p> <p>The recommended value is a range of bitrates, in bits per second. For example, "0-2427000".</p> <p>Note: while you can use a specific bitrate value, like 250000 (bits per second), this approach is not recommended, as the exact bitrates can fluctuate from one Asset to another.</p>
FourCC	<p>Use the FourCC value of the track for filtering.</p> <p>The value is the first element of codecs format, as specified in RFC 6381. Currently, the following codecs are supported: For Video: "avc1", "hev1", "hvc1" For Audio: "mp4a", "ec-3"</p> <p>To determine the FourCC values for tracks in an Asset, get and examine the manifest file.</p>
Language	<p>Use the language of the track for filtering.</p> <p>The value is the tag of a language you want to include, as specified in RFC 5646. For example, "en".</p>
Name	Use the name of the track for filtering.
Type	<p>Use the type of the track for filtering.</p> <p>The following values are allowed: "video", "audio", or "text".</p>

Example

```

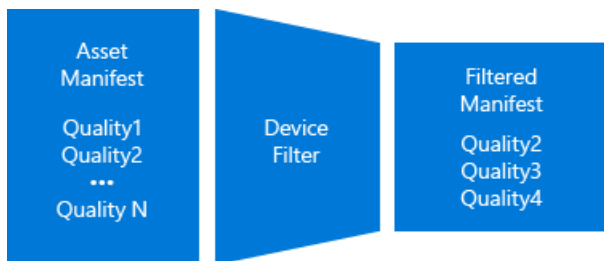
{
  "properties": {
    "presentationTimeRange": {
      "startTimestamp": 0,
      "endTimestamp": 170000000,
      "presentationWindowDuration": 9223372036854776000,
      "liveBackoffDuration": 0,
      "timescale": 10000000,
      "forceEndTimestamp": false
    },
    "firstQuality": {
      "bitrate": 128000
    },
    "tracks": [
      {
        "trackSelections": [
          {
            "property": "Type",
            "operation": "Equal",
            "value": "Audio"
          },
          {
            "property": "Language",
            "operation": "NotEqual",
            "value": "en"
          },
          {
            "property": "FourCC",
            "operation": "NotEqual",
            "value": "EC-3"
          }
        ]
      },
      {
        "trackSelections": [
          {
            "property": "Type",
            "operation": "Equal",
            "value": "Video"
          },
          {
            "property": "Bitrate",
            "operation": "Equal",
            "value": "3000000-5000000"
          }
        ]
      }
    ]
  }
}

```

Rendition filtering

You may choose to encode your asset to multiple encoding profiles (H.264 Baseline, H.264 High, AAC-L, AAC-H, Dolby Digital Plus) and multiple quality bitrates. However, not all client devices will support all your asset's profiles and bitrates. For example, older Android devices only support H.264 Baseline+AAC-L. Sending higher bitrates to a device, which cannot get the benefits, wastes bandwidth, and device computation. Such device must decode all the given information, only to scale it down for display.

With Dynamic Manifest, you can create device profiles such as mobile, console, HD/SD, etc. and include the tracks and qualities, which you want to be a part of each profile.



In the following example, an encoder was used to encode a mezzanine asset into seven ISO MP4s video renditions (from 180p to 1080p). The encoded asset can be dynamically packaged into any of the following streaming protocols: HLS, MPEG DASH, and Smooth. At the top of the diagram, the HLS manifest for the asset with no filters is shown (it contains all seven renditions). In the bottom left, the HLS manifest to which a filter named "ott" was applied is shown. The "ott" filter specifies to remove all bitrates below 1 Mbps, which resulted in the bottom two quality levels being stripped off in the response. In the bottom right, the HLS manifest to which a filter named "mobile" was applied is shown. The "mobile" filter specifies to remove renditions where the resolution is larger than 720p, which resulted in the two 1080p renditions being stripped off.

[http://.../asset.ism/manifest\(format=m3u8-aapl-v3\)](http://.../asset.ism/manifest(format=m3u8-aapl-v3))

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-STREAM-INF:BANDWIDTH=593669,RESOLUTION=320x180,CODECS="avc1.64000d,mp4a.40.2"
QualityLevels(439486)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=872173,RESOLUTION=640x360,CODECS="avc1.64001e,mp4a.40.2"
QualityLevels(711995)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=1264329,RESOLUTION=640x360,CODECS="avc1.64001e,mp4a.40.2"
QualityLevels(1095709)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=1817959,RESOLUTION=960x540,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(1637421)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=2650329,RESOLUTION=960x540,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(2451873)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=3927545,RESOLUTION=1280x720,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(3701595)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=5346983,RESOLUTION=1920x1080,CODECS="avc1.640028,mp4a.40.2"
QualityLevels(5090478)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=6783728,RESOLUTION=1920x1080,CODECS="avc1.640028,mp4a.40.2"
QualityLevels(6496295)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=62909,CODECS="mp4a.40.2"
QualityLevels(53571)/Manifest(AAC_and_ch2_56kbps,format=m3u8-aapl-v3)
```

[http://.../asset.ism/manifest\(format=m3u8-aapl-v3,filter=ott\)](http://.../asset.ism/manifest(format=m3u8-aapl-v3,filter=ott))

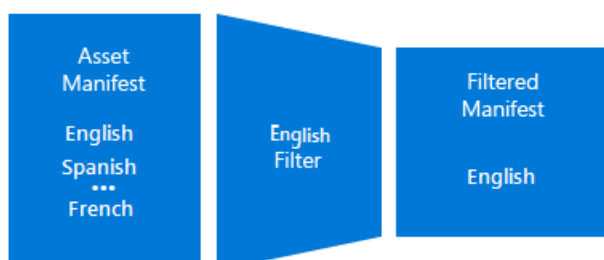
```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-STREAM-INF:BANDWIDTH=1264329,RESOLUTION=640x360,CODECS="avc1.64001e,mp4a.40.2"
QualityLevels(1095709)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=1817959,RESOLUTION=960x540,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(1637421)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=2650329,RESOLUTION=960x540,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(2451873)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=3927545,RESOLUTION=1280x720,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(3701595)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=5346983,RESOLUTION=1920x1080,CODECS="avc1.640028,mp4a.40.2"
QualityLevels(5090478)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=6783728,RESOLUTION=1920x1080,CODECS="avc1.640028,mp4a.40.2"
QualityLevels(6496295)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=62909,CODECS="mp4a.40.2"
QualityLevels(53571)/Manifest(AAC_and_ch2_56kbps,format=m3u8-aapl-v3)
```

[http://.../asset.ism/manifest\(format=m3u8-aapl-v3,filter=mobile\)](http://.../asset.ism/manifest(format=m3u8-aapl-v3,filter=mobile))

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-STREAM-INF:BANDWIDTH=593669,RESOLUTION=320x180,CODECS="avc1.64000d,mp4a.40.2"
QualityLevels(439486)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=872173,RESOLUTION=640x360,CODECS="avc1.64001e,mp4a.40.2"
QualityLevels(711995)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=1264329,RESOLUTION=640x360,CODECS="avc1.64001e,mp4a.40.2"
QualityLevels(1095709)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=1817959,RESOLUTION=960x540,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(1637421)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=2650329,RESOLUTION=960x540,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(2451873)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=3927545,RESOLUTION=1280x720,CODECS="avc1.64001f,mp4a.40.2"
QualityLevels(3701595)/Manifest(video,format=m3u8-aapl-v3,audiotrack=AAC_and_ch2_56kbps)
#EXT-X-STREAM-INF:BANDWIDTH=62909,CODECS="mp4a.40.2"
QualityLevels(53571)/Manifest(AAC_and_ch2_56kbps,format=m3u8-aapl-v3)
```

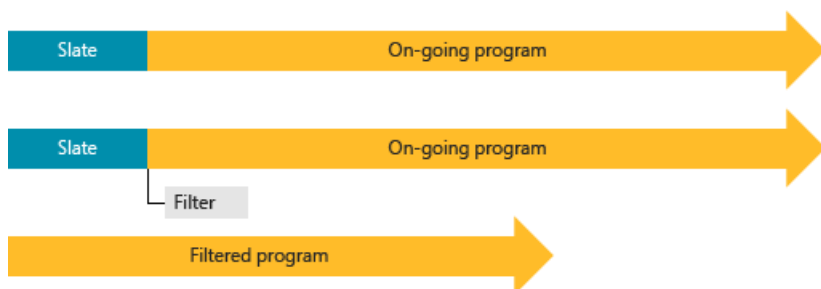
Removing language tracks

Your assets might include multiple audio languages such as English, Spanish, French, etc. Usually, the Player SDK managers default audio track selection and available audio tracks per user selection. It is challenging to develop such Player SDKs, it requires different implementations across device-specific player-frameworks. Also, on some platforms, Player APIs are limited and do not include audio selection feature where users cannot select or change the default audio track. With asset filters, you can control the behavior by creating filters that only include desired audio languages.



Trimming start of an asset

In most live streaming events, operators run some tests before the actual event. For example, they might include a slate like this before the start of the event: "Program will begin momentarily". If the program is archiving, the test and slate data are also archived and included in the presentation. However, this information should not be shown to the clients. With Dynamic Manifest, you can create a start time filter and remove the unwanted data from the manifest.



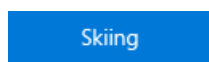
Creating subclips (views) from a live archive

Many live events are long running and live archive might include multiple events. After the live event ends, broadcasters may want to break up the live archive into logical program start and stop sequences. Next, publish these virtual programs separately without post processing the live archive and not creating separate assets (which does not get the benefit of the existing cached fragments in the CDNs). Examples of such virtual programs are the quarters of a football or basketball game, innings in baseball, or individual events of any sports program.

With Dynamic Manifest, you can create filters using start/end times and create virtual views over the top of your live archive.



Filtered Asset:



Adjusting Presentation Window (DVR)

Currently, Azure Media Services offers circular archive where the duration can be configured between 5 minutes - 25 hours. Manifest filtering can be used to create a rolling DVR window over the top of the archive, without deleting media. There are many scenarios where broadcasters want to provide a limited DVR window to move with the live edge and at the same time keep a bigger archiving window. A broadcaster may want to use the data that is out of the DVR window to highlight clips, or they may want to provide different DVR windows for different devices. For example, most of the mobile devices don't handle large DVR windows (you can have a 2-minute DVR window for mobile devices and one hour for desktop clients).



Adjusting LiveBackoff (live position)

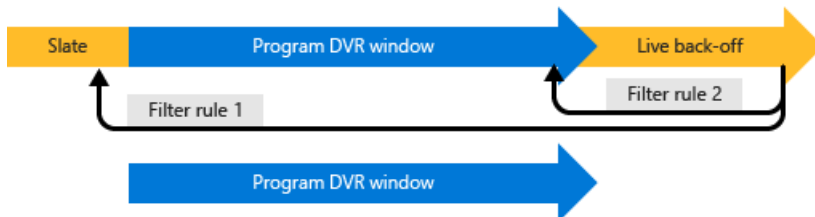
Manifest filtering can be used to remove several seconds from the live edge of a live program. Filtering allows broadcasters to watch the presentation on the preview publication point and create advertisement insertion points before the viewers receive the stream (backed-off by 30 seconds). Broadcasters can then push these advertisements to their client frameworks in time for them to receive and process the information before the advertisement opportunity.

In addition to the advertisement support, the LiveBackoff setting can be used to adjusting the viewers position so that when clients drift and hit the live edge they can still get fragments from server instead of getting an HTTP 404 or 412 error.



Combining multiple rules in a single filter

You can combine multiple filtering rules in a single filter. As an example you can define a "range rule" to remove slates from a live archive and also filter out available bitrates. When applying multiple filtering rules, the end result is the intersection of all rules.



Combining multiple filters (filter composition)

You can also combine multiple filters in a single URL.

The following scenario demonstrates why you might want to combine filters:

1. You need to filter your video qualities for mobile devices such as Android or iPad (in order to limit video qualities). To remove the unwanted qualities, you would create an Account filter suitable for the device profiles. Account filters can be used for all your assets under the same media services account without any further association.
2. You also want to trim the start and end time of an asset. To achieve this, you would create an Asset filter and set the start/end time.
3. You want to combine both of these filters (without combination, you need to add quality filtering to the trimming filter, which makes filter usage more difficult).

To combine filters, you need to set the filter names to the manifest/playlist URL with semicolon delimited. Let's assume you have a filter named *MyMobileDevice* that filters qualities and you have another named *MyStartTime* to set a specific start time. You can combine them like this:

You can combine up to three filters.

For more information, see [this](#) blog.

Considerations and limitations

- The values for **forceEndTimestamp**, **presentationWindowDuration**, and **liveBackoffDuration** should not be set for a VoD filter. They are only used for live filter scenarios.
- Dynamic manifest operates in GOP boundaries (Key Frames) hence trimming has GOP accuracy.
- You can use same filter name for Account and Asset filters. Asset filters have higher precedence and will override Account filters.
- If you update a filter, it can take up to 2 minutes for streaming endpoint to refresh the rules. If the content was served using some filters (and cached in proxies and CDN caches), updating these filters can result in player failures. It is recommended to clear the cache after updating the filter. If this option is not possible, consider using a different filter.

- Customers need to manually download the manifest and parse the exact startTimestamp and time scale.
 - To determine properties of the tracks in an Asset, [get and examine the manifest file](#).
 - The formula to set the asset filter timestamp properties:
$$\text{startTimestamp} = \text{<start time in the manifest>} + \text{<expected filter start time in seconds>} * \text{timescale}$$

Next steps

The following articles show how to create filters programmatically.

- [Create filters with REST APIs](#)
- [Create filters with .NET](#)
- [Create filters with CLI](#)

Assets

12/18/2018 • 6 minutes to read • [Edit Online](#)

In Azure Media Services, an [Asset](#) contains digital files (including video, audio, images, thumbnail collections, text tracks and closed caption files) and the metadata about these files. After the digital files are uploaded into an Asset, they can be used in the Media Services encoding, streaming, analyzing content workflows. For more information, see the [Upload digital files into Assets](#) section below.

An Asset is mapped to a blob container in the [Azure Storage account](#) and the files in the Asset are stored as block blobs in that container. Media Services supports Blob tiers when the account uses General-purpose v2 (GPv2) storage. With GPv2, you can move files to [Cool or Archive storage](#). **Archive** storage is suitable for archiving source files when no longer needed (for example, after they have been encoded).

The **Archive** storage tier is only recommended for very large source files that have already been encoded and the encoding Job output was put in an output blob container. The blobs in the output container that you want to associate with an Asset and use to stream or analyze your content, must exist in a **Hot** or **Cool** storage tier.

Asset definition

The following table shows the Asset's properties and gives their definitions.

NAME	DESCRIPTION
id	Fully qualified resource ID for the resource.
name	The name of the resource.
properties.alternateId	The alternate ID of the Asset.
properties.assetId	The Asset ID.
properties.container	The name of the asset blob container.
properties.created	The creation date of the Asset. Datetime is always in UTC format.
properties.description	The Asset description.
properties.lastModified	The last modified date of the Asset. Datetime is always in UTC format.
properties.storageAccountName	The name of the storage account.
properties.storageEncryptionFormat	The Asset encryption format. One of None or MediaStorageEncryption.
type	The type of the resource.

For a full definition, see [Assets](#).

Upload digital files into Assets

One of the common Media Services workflows is to upload, encode, and stream a file. This section outlines the general steps.

1. Use the Media Services v3 API to create a new "input" Asset. This operation creates a container in the storage account associated with your Media Services account. The API returns the container name (for example, `"container": "asset-b8d8b68a-2d7f-4d8c-81bb-8c7bbe67ee4"`).

If you already have a blob container that you want to associate with an Asset, you can specify the container name when creating the Asset. Media Services currently only supports blobs in the container root and not with paths in the file name. Thus, a container with the "input.mp4" file name will work. However, a container with the "videos/inputs/input.mp4" file name, will not work.

You can use the Azure CLI to upload directly to any storage account and container that you have rights to in your subscription.

The container name must be unique and follow storage naming guidelines. The name doesn't have to follow the Media Services Asset container name (Asset-GUID) formatting.

```
az storage blob upload -f /path/to/file -c MyContainer -n MyBlob
```

2. Get a SAS URL with read-write permissions that will be used to upload digital files into the Asset container. You can use the Media Services API to [list the asset container URLs](#).
3. Use the Azure Storage APIs or SDKs (for example, the [Storage REST API](#), [JAVA SDK](#), or [.NET SDK](#)) to upload files into the Asset container.
4. Use Media Services v3 APIs to create a Transform and a Job to process your "input" Asset. For more information, see [Transforms and Jobs](#).
5. Stream the content from the "output" asset.

TIP

For a full .NET example that shows how to: create the Asset, get a writable SAS URL to the Asset's container in storage, upload the file into the container in storage using the SAS URL, see [Create a job input from a local file](#).

Create a new asset

REST

```
PUT
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Media/mediaServices/{amsAccountName}/assets/{assetName}?api-version=2018-07-01
```

For a REST example, see the [Create an Asset with REST](#) example.

The example shows how to create the **Request Body** where you can specify useful information like description, container name, storage account, and other information.

cURL

```
curl -X PUT \
  'https://management.azure.com/subscriptions/00000000-0000-0000-000000000000/resourceGroups/resourceGroupName/providers/Microsoft.Media/mediaServices/amsAccountName/assets/myOutputAsset?api-version=2018-07-01' \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "properties": {
      "description": "",
    }
  }'
```

.NET

```
Asset asset = await client.Assets.CreateOrUpdateAsync(resourceGroupName, accountName, assetName, new Asset());
```

For a full example, see [Create a job input from a local file](#). In Media Services v3, a job's input can also be created from HTTPS URLs (see [Create a job input from an HTTPS URL](#)).

Filtering, ordering, paging

Media Services supports the following OData query options for Assets:

- \$filter
- \$orderby
- \$top
- \$skiptoken

Operator description:

- Eq = equal to
- Ne = not equal to
- Ge = Greater than or equal to
- Le = Less than or equal to
- Gt = Greater than
- Lt = Less than

Filtering/ordering

The following table shows how these options may be applied to the Asset properties:

NAME	FILTER	ORDER
id		
name	Supports: Eq, Gt, Lt	Supports: Ascending and Descending
properties.alternateId	Supports: Eq	
properties.assetId	Supports: Eq	
properties.container		
properties.created	Supports: Eq, Gt, Lt	Supports: Ascending and Descending

NAME	FILTER	ORDER
properties.description		
properties.lastModified		
properties.storageAccountName		
properties.storageEncryptionFormat		
type		

The following C# example filters on the created date:

```
var odataQuery = new ODataQuery<Asset>("properties/created lt 2018-05-11T17:39:08.387Z");
var firstPage = await MediaServicesArmClient.Assets.ListAsync(CustomerResourceGroup, CustomerAccountName,
odataQuery);
```

Pagination

Pagination is supported for each of the four enabled sort orders. Currently, the page size is 1000.

TIP

You should always use the next link to enumerate the collection and not depend on a particular page size.

If a query response contains many items, the service returns an "@odata.nextLink" property to get the next page of results. This can be used to page through the entire result set. You cannot configure the page size.

If Assets are created or deleted while paging through the collection, the changes are reflected in the returned results (if those changes are in the part of the collection that has not been downloaded).

C# example

The following C# example shows how to enumerate through all the assets in the account.

```
var firstPage = await MediaServicesArmClient.Assets.ListAsync(CustomerResourceGroup, CustomerAccountName);

var currentPage = firstPage;
while (currentPage.NextPageLink != null)
{
    currentPage = await MediaServicesArmClient.Assets.ListNextAsync(currentPage.NextPageLink);
}
```

REST example

Consider the following example of where \$skiptoken is used. Make sure you replace *amstestaccount* with your account name and set the *api-version* value to the latest version.

If you request a list of Assets like this:

```
GET https://management.azure.com/subscriptions/00000000-3761-485c-81bb-
c50b291ce214/resourceGroups/mediaresources/providers/Microsoft.Media/mediaServices/amstestaccount/assets?api-
version=2018-07-01 HTTP/1.1
x-ms-client-request-id: dd57fe5d-f3be-4724-8553-4ceb1dbe5aab
Content-Type: application/json; charset=utf-8
```

You would get back a response similar to this:

HTTP/1.1 200 OK

```
{
  "value": [
    {
      "name": "Asset 0", "id": "/subscriptions/00000000-3761-485c-81bb-c50b291ce214/resourceGroups/mediaresources/providers/Microsoft.Media/mediaservices/amstestaccount/assets/Asset 0", "type": "Microsoft.Media/mediaservices/assets", "properties": {
        "assetId": "00000000-5a4f-470a-9d81-6037d7c23eff", "created": "2018-12-11T22:12:44.98Z", "lastModified": "2018-12-11T22:15:48.003Z", "container": "asset-98d07299-5a4f-470a-9d81-6037d7c23eff", "storageAccountName": "amsdevc1stoaccount11", "storageEncryptionFormat": "None"
      }
    },
    // lots more assets
    {
      "name": "Asset 517", "id": "/subscriptions/00000000-3761-485c-81bb-c50b291ce214/resourceGroups/mediaresources/providers/Microsoft.Media/mediaservices/amstestaccount/assets/Asset 517", "type": "Microsoft.Media/mediaservices/assets", "properties": {
        "assetId": "00000000-912e-447b-a1ed-0f723913b20d", "created": "2018-12-11T22:14:08.473Z", "lastModified": "2018-12-11T22:19:29.657Z", "container": "asset-fd05a503-912e-447b-a1ed-0f723913b20d", "storageAccountName": "amsdevc1stoaccount11", "storageEncryptionFormat": "None"
      }
    }
  ], "@odata.nextLink": "https://management.azure.com/subscriptions/00000000-3761-485c-81bb-c50b291ce214/resourceGroups/mediaresources/providers/Microsoft.Media/mediaServices/amstestaccount/assets?api-version=2018-07-01&$skiptoken=Asset+517"
}
```

You would then request the next page by sending a get request for:

```
https://management.azure.com/subscriptions/00000000-3761-485c-81bb-c50b291ce214/resourceGroups/mediaresources/providers/Microsoft.Media/mediaServices/amstestaccount/assets?api-version=2018-07-01&$skiptoken=Asset+517
```

For more REST examples, see [Assets - List](#)

Storage side encryption

To protect your Assets at rest, the assets should be encrypted by the storage side encryption. The following table shows how the storage side encryption works in Media Services:

ENCRYPTION OPTION	DESCRIPTION	MEDIA SERVICES V2	MEDIA SERVICES V3
Media Services Storage Encryption	AES-256 encryption, key managed by Media Services	Supported ⁽¹⁾	Not supported ⁽²⁾
Storage Service Encryption for Data at Rest	Server-side encryption offered by Azure Storage, key managed by Azure or by customer	Supported	Supported
Storage Client-Side Encryption	Client-side encryption offered by Azure storage, key managed by customer in Key Vault	Not supported	Not supported

¹ While Media Services does support handling of content in the clear/without any form of encryption, doing so is not recommended.

² In Media Services v3, storage encryption (AES-256 encryption) is only supported for backwards compatibility

when your Assets were created with Media Services v2. Meaning v3 works with existing storage encrypted assets but will not allow creation of new ones.

Next steps

[Stream a file](#)

Content Key Policies

12/10/2018 • 2 minutes to read • [Edit Online](#)

You can use Azure Media Services to secure your media from the time it leaves your computer through storage, processing, and delivery. With Media Services, you can deliver your live and on-demand content encrypted dynamically with Advanced Encryption Standard (AES-128) or any of the three major digital rights management (DRM) systems: Microsoft PlayReady, Google Widevine, and Apple FairPlay. Media Services also provides a service for delivering AES keys and DRM (PlayReady, Widevine, and FairPlay) licenses to authorized clients.

In Azure Media Services v3, Content Key Policies enable you to specify how the content key is delivered to end clients via the Media Services Key Delivery component. For more information, see [Content protection overview](#).

ContentKeyPolicies definition

The following table shows the ContentKeyPolicy's properties and gives their definitions.

NAME	DESCRIPTION
id	Fully qualified resource ID for the resource.
name	The name of the resource.
properties.created	The creation date of the Policy
properties.description	A description for the Policy.
properties.lastModified	The last modified date of the Policy
properties.options	The Key Policy options.
properties.policyId	The legacy Policy ID.
type	The type of the resource.

For the full definition, see [Content Key Policies](#).

Filtering, ordering, paging

Media Services supports the following OData query options for ContentKeyPolicies:

- \$filter
- \$orderby
- \$top
- \$skiptoken

Operator description:

- Eq = equal to
- Ne = not equal to
- Ge = Greater than or equal to

- Le = Less than or equal to
- Gt = Greater than
- Lt = Less than

Filtering/ordering

The following table shows how these options may be applied to the StreamingPolicy properties:

NAME	FILTER	ORDER
id		
name	Eq, ne, ge, le, gt, lt	Ascending and descending
properties.created	Eq, ne, ge, le, gt, lt	Ascending and descending
properties.description	Eq, ne, ge, le, gt, lt	
properties.lastModified	Eq, ne, ge, le, gt, lt	Ascending and descending
properties.options		
properties.policyId	Eq, ne	
type		

Pagination

Pagination is supported for each of the four enabled sort orders. Currently, the page size is 10.

TIP

You should always use the next link to enumerate the collection and not depend on a particular page size.

If a query response contains many items, the service returns an "@odata.nextLink" property to get the next page of results. This can be used to page through the entire result set. You cannot configure the page size.

If StreamingPolicy are created or deleted while paging through the collection, the changes are reflected in the returned results (if those changes are in the part of the collection that has not been downloaded.)

The following C# example shows how to enumerate through all ContentKeyPolicies in the account.

```
var firstPage = await MediaServicesArmClient.ContentKeyPolicies.ListAsync(CustomerResourceGroup,
CustomerAccountName);

var currentPage = firstPage;
while (currentPage.NextPageLink != null)
{
    currentPage = await MediaServicesArmClient.ContentKeyPolicies.ListNextAsync(currentPage.NextPageLink);
}
```

For REST examples, see [Content Key Policies - List](#)

Next steps

[Use AES-128 dynamic encryption and the key delivery service](#)

Use DRM dynamic encryption and license delivery service

Streaming Endpoints

11/7/2018 • 7 minutes to read • [Edit Online](#)

In Microsoft Azure Media Services (AMS), the [Streaming Endpoints](#) entity represents a streaming service that can deliver content directly to a client player application, or to a Content Delivery Network (CDN) for further distribution. The outbound stream from a Streaming Endpoint service can be a live stream, or a video on-demand Asset in your Media Services account. When you create a Media Services account, a **default** Streaming Endpoint is created for you in a stopped state. You cannot delete the default Streaming Endpoint. Additional Streaming Endpoints can be created under the account. To start streaming videos, you need to start the Streaming Endpoint.

StreamingEndpoint types

There are two **StreamingEndpoint** types: **Standard** and **Premium**. The type is defined by the number of scale units (`scaleUnits`) you allocate for the streaming endpoint.

The table describes the types:

TYPE	SCALE UNITS	DESCRIPTION
Standard Streaming Endpoint (recommended)	0	The Standard type is the recommended option for virtually all streaming scenarios and audience sizes. The Standard type scales outbound bandwidth automatically. For customers with extremely demanding requirements Media Services offers Premium streaming endpoints, which can be used to scale out capacity for the largest internet audiences. If you expect large audiences and concurrent viewers, contact us at amsstreaming@microsoft.com for guidance on whether you need to move to the Premium type.
Premium Streaming Endpoint	>0	Premium streaming endpoints are suitable for advanced workloads, providing dedicated and scalable bandwidth capacity. You move to a Premium type by adjusting <code>scaleUnits</code> . <code>scaleUnits</code> provide you with dedicated egress capacity that can be purchased in increments of 200 Mbps. When using the Premium type, each enabled unit provides additional bandwidth capacity to the application.

Working with CDN

In most cases, you should have CDN enabled. However, if you are anticipating max concurrency lower than 500 viewers then it is recommended to disable CDN since CDN scales best with concurrency.

Detailed explanation of how caching works

There is no specific bandwidth value when adding the CDN because the amount of bandwidth that is needed for a

CDN enabled streaming endpoint varies. A lot depends on the type of content, how popular it is, bitrates, and the protocols. The CDN is only caching what is being requested. That means that popular content will be served directly from the CDN – as long as the video fragment is cached. Live content is likely to be cached because you typically have many people watching the exact same thing. On-demand content can be a bit trickier because you could have some content that is popular and some that is not. If you have millions of video assets where none of them are popular (only 1 or 2 viewers a week) but you have thousands of people watching all different videos, the CDN becomes much less effective. With this cache misses, you increase the load on the streaming endpoint.

You also need to consider how adaptive streaming works. Each individual video fragment is cached as it's own entity. For example, if the first time a certain video is watched, the person skips around watching only a few seconds here and there only the video fragments associated with what the person watched get cached in the CDN. With adaptive streaming, you typically have 5 to 7 different bitrates of video. If one person is watching one bitrate and another person is watching a different bitrate, then they are each cached separately in the CDN. Even if two people are watching the same bitrate they could be streaming over different protocols. Each protocol (HLS, MPEG-DASH, Smooth Streaming) is cached separately. So each bitrate and protocol are cached separately and only those video fragments that have been requested are cached.

StreamingEndpoint properties

This section gives details about some of the StreamingEndpoint's properties. For examples of how to create a new streaming endpoint and descriptions of all properties, see [Streaming Endpoint](#).

PROPERTY	DESCRIPTION
<code>accessControl</code>	<p>Used to configure the following security settings for this streaming endpoint: Akamai signature header authentication keys and IP addresses that are allowed to connect to this endpoint.</p> <p>This property can be set when <code>cdnEnabled</code> is set to false.</p>
<code>cdnEnabled</code>	<p>Indicates whether or not the Azure CDN integration for this streaming endpoint is enabled (disabled by default.)</p> <p>If you set <code>cdnEnabled</code> to true, the following configurations get disabled: <code>customHostNames</code> and <code>accessControl</code>.</p> <p>Not all data centers support the Azure CDN integration. To check whether or not your data center has the Azure CDN integration available do the following:</p> <ul style="list-style-type: none">- Try to set the <code>cdnEnabled</code> to true.- Check the returned result for an <code>HTTP Error Code 412</code> (PreconditionFailed) with a message of "Streaming endpoint CdnEnabled property cannot be set to true as CDN capability is not available in the current region." <p>If you get this error, the data center does not support it. You should try another data center.</p>

PROPERTY	DESCRIPTION
<code>cdnProfile</code>	<p>When <code>cdnEnabled</code> is set to true, you can also pass <code>cdnProfile</code> values. <code>cdnProfile</code> is the name of the CDN profile where the CDN endpoint point will be created. You can provide an existing <code>cdnProfile</code> or use a new one. If value is NULL and <code>cdnEnabled</code> is true, the default value "AzureMediaStreamingPlatformCdnProfile" is used. If the provided <code>cdnProfile</code> already exists, an endpoint is created under it. If the profile does not exists, a new profile automatically gets created.</p>
<code>cdnProvider</code>	<p>When CDN is enabled, you can also pass <code>cdnProvider</code> values. <code>cdnProvider</code> controls which provider will be used. Currently, three values are supported: "StandardVerizon", "PremiumVerizon" and "StandardAkamai". If no value is provided and <code>cdnEnabled</code> is true, "StandardVerizon" is used (that is the default value.)</p>
<code>crossSiteAccessPolicies</code>	<p>Used to specify cross site access policies for various clients. For more information, see Cross-domain policy file specification and Making a Service Available Across Domain Boundaries.</p>
<code>customHostNames</code>	<p>Used to configure a streaming endpoint to accept traffic directed to a custom host name. This allows for easier traffic management configuration through a Global Traffic Manager (GTM) and also for branded domain names to be used as the streaming endpoint name.</p> <p>The ownership of the domain name must be confirmed by Azure Media Services. Azure Media Services verifies the domain name ownership by requiring a <code>CName</code> record containing the Azure Media Services account ID as a component to be added to the domain in use. As an example, for "sports.contoso.com" to be used as a custom host name for the streaming endpoint, a record for <code><accountId>.contoso.com</code> must be configured to point to one of Media Services verification host names. The verification host name is composed of <code>verifydns.<mediaservices-dns-zone></code>. The following table contains the expected DNS zones to be used in the verify record for different Azure regions.</p> <p>North America, Europe, Singapore, Hong Kong, Japan:</p> <ul style="list-style-type: none"> - mediaservices.windows.net - verifydns.mediaservices.windows.net <p>China:</p> <ul style="list-style-type: none"> - mediaservices.chinacloudapi.cn - verifydns.mediaservices.chinacloudapi.cn <p>For example, a <code>CName</code> record that maps "945a4c4e-28ea-45cd-8ccb-a519f6b700ad.contoso.com" to "verifydns.mediaservices.windows.net" proves that the Azure Media Services ID 945a4c4e-28ea-45cd-8ccb-a519f6b700ad has the ownership of the contoso.com domain, thus enabling any name under contoso.com to be used as a custom host name for a streaming endpoint under that account.</p>

PROPERTY	DESCRIPTION
	<p>To find the Media Service ID value, go to the Azure portal and select your Media Service account. The MEDIA SERVICE ID appears on the right of the DASHBOARD page.</p> <p>Warning: If there is an attempt to set a custom host name without a proper verification of the <code>CName</code> record, the DNS response will fail and then be cached for some time. Once a proper record is in place, it might take a while until the cached response is revalidated. Depending on the DNS provider for the custom domain, it could take anywhere from a few minutes to an hour to revalidate the record.</p> <p>In addition to the <code>CName</code> that maps <code><accountId>.<parent domain></code> to <code>verifydns.<mediaservices-dns-zone></code>, you must create another <code>CName</code> that maps the custom host name (for example, <code>sports.contoso.com</code>) to your Media Services StreamingEndpoint's host name (for example, <code>amstest.streaming.mediaservices.windows.net</code>).</p> <p>Note: Streaming endpoints located in the same data center, cannot share the same custom host name. This property is valid for Standard and premium streaming endpoints and can be set when "cdnEnabled": false</p> <p>Currently, AMS doesn't support SSL with custom domains.</p>
<code>maxCacheAge</code>	Overrides the default max-age HTTP cache control header set by the streaming endpoint on media fragments and on-demand manifests. The value is set in seconds.
<code>resourceState</code>	<p>Values for the property include:</p> <ul style="list-style-type: none"> - Stopped. The initial state of a streaming endpoint after creation. - Starting. The streaming endpoint is transitioning to the running state. - Running. The streaming endpoint is able to stream content to clients. - Scaling. The scale units are being increased or decreased. - Stopping. The streaming endpoint is transitioning to the stopped state. - Deleting. The streaming endpoint is being deleted.
<code>scaleUnits</code>	scaleUnits provide you with dedicated egress capacity that can be purchased in increments of 200 Mbps. If you need to move to a Premium type, adjust <code>scaleUnits</code> .

Next steps

The sample [in this repository](#) shows how to start the default streaming endpoint with .NET.

Streaming Locators

10/25/2018 • 2 minutes to read • [Edit Online](#)

To provide your clients with a URL that they can use to play back encoded video or audio files, you need to create a [StreamingLocator](#) and build the streaming URLs. For more information, see [Stream a file](#).

StreamingLocator definition

The following table shows the StreamingLocator's properties and gives their definitions.

NAME	DESCRIPTION
id	Fully qualified resource ID for the resource.
name	The name of the resource.
properties.alternativeMediaId	Alternative Media ID of this Streaming Locator.
properties.assetName	Asset name
properties.contentKeys	The ContentKeys used by this Streaming Locator.
properties.created	The creation time of the Streaming Locator.
properties.defaultContentKeyPolicyName	Name of the default ContentKeyPolicy used by this Streaming Locator.
properties.endTime	The end time of the Streaming Locator.
properties.startTime	The start time of the Streaming Locator.
properties.streamingLocatorId	The StreamingLocatorId of the Streaming Locator.
properties.streamingPolicyName	Name of the Streaming Policy used by this Streaming Locator. Either specify the name of Streaming Policy you created or use one of the predefined Streaming Policies. The predefined Streaming Policies available are: 'Predefined_DownloadOnly', 'Predefined_ClearStreamingOnly', 'Predefined_DownloadAndClearStreaming', 'Predefined_ClearKey', 'Predefined_MultiDrmCencStreaming' and 'Predefined_MultiDrmStreaming'
type	The type of the resource.

For the full definition, see [Streaming Locators](#).

Filtering, ordering, paging

Media Services supports the following OData query options for Streaming Locators:

- `$filter`

- \$orderby
- \$top
- \$skiptoken

Operator description:

- Eq = equal to
- Ne = not equal to
- Ge = Greater than or equal to
- Le = Less than or equal to
- Gt = Greater than
- Lt = Less than

Filtering/ordering

The following table shows how these options may be applied to the StreamingLocator properties:

NAME	FILTER	ORDER
id		
name	Eq, ne, ge, le, gt, lt	Ascending and descending
properties.alternativeMediaId		
properties.assetName		
properties.contentKeys		
properties.created	Eq, ne, ge, le, gt, lt	Ascending and descending
properties.defaultContentKeyPolicyName		
properties.endTime	Eq, ne, ge, le, gt, lt	Ascending and descending
properties.startTime		
properties.streamingLocatorId		
properties.streamingPolicyName		
type		

Pagination

Pagination is supported for each of the four enabled sort orders. Currently, the page size is 10.

TIP

You should always use the next link to enumerate the collection and not depend on a particular page size.

If a query response contains many items, the service returns an "@odata.nextLink" property to get the next page of results. This can be used to page through the entire result set. You cannot configure the page size.

If StreamingLocators are created or deleted while paging through the collection, the changes are reflected in the returned results (if those changes are in the part of the collection that has not been downloaded.)

The following C# example shows how to enumerate through all StreamingLocators in the account.

```
var firstPage = await MediaServicesArmClient.StreamingLocators.ListAsync(CustomerResourceGroup,
CustomerAccountName);

var currentPage = firstPage;
while (currentPage.NextPageLink != null)
{
    currentPage = await MediaServicesArmClient.StreamingLocators.ListNextAsync(currentPage.NextPageLink);
}
```

For REST examples, see [Streaming Locators - List](#)

Next steps

[Stream a file](#)

Streaming Policies

10/25/2018 • 2 minutes to read • [Edit Online](#)

In Azure Media Services v3, Streaming Policies enable you to define streaming protocols and encryption options for your StreamingLocators. You can either specify the name of Streaming Policy you created or use one of the predefined Streaming Policies. The predefined Streaming Policies currently available are:

'Predefined_DownloadOnly', 'Predefined_ClearStreamingOnly', 'Predefined_DownloadAndClearStreaming', 'Predefined_ClearKey', 'Predefined_MultiDrmCencStreaming' and 'Predefined_MultiDrmStreaming'.

IMPORTANT

When using a custom [StreamingPolicy](#), you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed. Your Media Service account has a quota for the number of StreamingPolicy entries. You should not be creating a new StreamingPolicy for each StreamingLocator.

StreamingPolicy definition

The following table shows the StreamingPolicy's properties and gives their definitions.

NAME	DESCRIPTION
id	Fully qualified resource ID for the resource.
name	The name of the resource.
properties.commonEncryptionCbcs	Configuration of CommonEncryptionCbcs
properties.commonEncryptionCenc	Configuration of CommonEncryptionCenc
properties.created	Creation time of Streaming Policy
properties.defaultContentKeyPolicyName	Default ContentKey used by current Streaming Policy
properties.envelopeEncryption	Configuration of EnvelopeEncryption
properties.noEncryption	Configurations of NoEncryption
type	The type of the resource.

For the full definition, see [Streaming Policies](#).

Filtering, ordering, paging

Media Services supports the following OData query options for Streaming Policies:

- \$filter
- \$orderby
- \$top

- \$skiptoken

Operator description:

- Eq = equal to
- Ne = not equal to
- Ge = Greater than or equal to
- Le = Less than or equal to
- Gt = Greater than
- Lt = Less than

Filtering/ordering

The following table shows how these options may be applied to the StreamingPolicy properties:

NAME	FILTER	ORDER
id		
name	Eq, ne, ge, le, gt, lt	Ascending and descending
properties.commonEncryptionCbcs		
properties.commonEncryptionCenc		
properties.created	Eq, ne, ge, le, gt, lt	Ascending and descending
properties.defaultContentKeyPolicyName		
properties.envelopeEncryption		
properties.noEncryption		
type		

Pagination

Pagination is supported for each of the four enabled sort orders. Currently, the page size is 10.

TIP

You should always use the next link to enumerate the collection and not depend on a particular page size.

If a query response contains many items, the service returns an "@odata.nextLink" property to get the next page of results. This can be used to page through the entire result set. You cannot configure the page size.

If StreamingPolicy are created or deleted while paging through the collection, the changes are reflected in the returned results (if those changes are in the part of the collection that has not been downloaded.)

The following C# example shows how to enumerate through all StreamingPolicies in the account.

```
var firstPage = await MediaServicesArmClient.StreamingPolicies.ListAsync(CustomerResourceGroup,
CustomerAccountName);

var currentPage = firstPage;
while (currentPage.NextPageLink != null)
{
    currentPage = await MediaServicesArmClient.StreamingPolicies.ListNextAsync(currentPage.NextPageLink);
}
```

For REST examples, see [Streaming Policies - List](#)

Next steps

[Stream a file](#)

Transforms and Jobs

10/24/2018 • 4 minutes to read • [Edit Online](#)

Azure Media Services v3 introduces a new templated workflow resource for a recipe that you want to use to encode and/or analyze your videos, called [Transforms](#). **Transforms** can be used to configure common tasks for encoding or analyzing videos. Each **Transform** describes a recipe, or a workflow of tasks for processing your video or audio files.

A **Job** is the actual request to Azure Media Services to apply the **Transform** to a given input video or audio content. The **Job** specifies information such as the location of the input video, and the location for the output. You can specify the location of your input video using: HTTPs URLs, SAS URLs, or [Media Services Assets](#).

Typical workflow

1. Create a Transform
2. Submit Jobs under that Transform
3. List Transforms
4. Delete a Transform, if you are not planning to use it in the future.

Suppose you wanted to extract the first frame of all your videos as a thumbnail image – the steps you would take are:

1. Define the recipe, or the rule for processing your videos - "use the first frame of the video as the thumbnail".
2. For each video you would tell the service:
 - a. Where to find that video,
 - b. Where to write the output thumbnail image.

A **Transform** helps you create the recipe once (Step 1), and submit Jobs using that recipe (Step 2).

Transforms

You can create Transforms in your Media Services account using the v3 API directly, or using any of the published SDKs. The Media Services v3 API is driven by Azure Resource Manager, so you can also use Resource Manager templates to create and deploy Transforms in your Media Services account. Role-based access control can be used to lock down access to Transforms.

Transform definition

The following table shows the Transform's properties and gives their definitions.

NAME	DESCRIPTION
Id	Fully qualified resource ID for the resource.
name	The name of the resource.
properties.created	The UTC date and time when the Transform was created, in 'YYYY-MM-DDThh:mm:ssZ' format.
properties.description	An optional verbose description of the Transform.

NAME	DESCRIPTION
properties.lastModified	The UTC date and time when the Transform was last updated, in 'YYYY-MM-DDThh:mm:ssZ' format.
properties.outputs	An array of one or more TransformOutputs that the Transform should generate.
type	The type of the resource.

For the full definition, see [Transforms](#).

As explained above, a Transform helps you create a recipe or rule for processing your videos. A single Transform can apply more than one rule. For example, a Transform could specify that each video be encoded into an MP4 file at a given bitrate, and that a thumbnail image be generated from the first frame of the video. You would add one TransformOutput entry for each rule that you want to include in your Transform.

NOTE

While the Transforms definition supports an Update operation, you should take care to make sure all in-progress Jobs complete before you make a modification. Typically, you would update an existing Transform to change the description, or modify the priorities of the underlying TransformOutputs. If you wanted to rewrite the recipe, then you would create a new Transform.

Jobs

Once a Transform has been created, you can submit Jobs using Media Services APIs, or any of the published SDKs. The progress and state of jobs can be obtained by monitoring events with Event Grid. For more information, see [Monitor events using EventGrid](#).

Jobs definition

The following table shows Jobs's properties and gives their definitions.

NAME	DESCRIPTION
id	Fully qualified resource ID for the resource.
name	The name of the resource.
properties.correlationData	Customer provided correlation data (immutable) that is returned as part of Job and JobOutput state change notifications. For more information, see Event schemas The property can also be used for multi-tenant usage of Media Services. You can put tenant IDs in the jobs.
properties.created	The UTC date and time when the Job was created, in 'YYYY-MM-DDThh:mm:ssZ' format.
properties.description	Optional customer supplied description of the Job.
properties.input	The inputs for the Job.

NAME	DESCRIPTION
properties.lastModified	The UTC date and time when the Job was last updated, in 'YYYY-MM-DDThh:mm:ssZ' format.
properties.outputs	The outputs for the Job.
properties.priority	Priority with which the job should be processed. Higher priority jobs are processed before lower priority jobs. If not set, the default is normal.
properties.state	The current state of the job.
type	The type of the resource.

For the full definition, see [Jobs](#).

NOTE

While the Jobs definition supports an Update operation, the only properties that can be modified after the Job is submitted are the description, and the priority. Further, a change to the priority is effective only if the Job is still in a queued state. If the job has begun processing, or has finished, changing the priority has no effect.

Pagination

Jobs pagination is supported in Media Services v3.

TIP

You should always use the next link to enumerate the collection and not depend on a particular page size.

If a query response contains many items, the service returns an "@odata.nextLink" property to get the next page of results. This can be used to page through the entire result set. You cannot configure the page size.

If Jobs are created or deleted while paging through the collection, the changes are reflected in the returned results (if those changes are in the part of the collection that has not been downloaded.)

The following C# example shows how to enumerate through the jobs in the account.


```
List<string> jobsToDelete = new List<string>();
var pageOfJobs = client.Jobs.List(config.ResourceGroup, config.AccountName, "Encode");

bool exit;
do
{
    foreach (Job j in pageOfJobs)
    {
        jobsToDelete.Add(j.Name);
    }

    if (pageOfJobs.NextPageLink != null)
    {
        pageOfJobs = client.Jobs.ListNext(pageOfJobs.NextPageLink);
        exit = false;
    }
    else
    {
        exit = true;
    }
}
while (!exit);
```

For REST examples, see [Jobs - List](#)

Next steps

[Stream video files](#)

Storage accounts

10/17/2018 • 2 minutes to read • [Edit Online](#)

When creating a Media Services account, you need to supply the name of an Azure Storage account resource. The specified storage account is attached to your Media Services account.

You must have one **Primary** storage account and you can have any number of **Secondary** storage accounts associated with your Media Services account. Media Services supports **General-purpose v2** (GPv2) or **General-purpose v1** (GPv1) accounts.

NOTE

Blob only accounts are not allowed as **Primary**.

We recommend that you use GPv2, so you can take advantage of choosing between hot and cool storage tiers. To learn more about storage accounts, see [Azure Storage account overview](#).

Assets in a storage account

In Media Services v3, the Storage APIs are used to upload files. To see how to use Storage APIs with Media Services to upload your input files, check out [Create a job input from a local file](#).

NOTE

You should not attempt to change the contents of blob containers that were generated by the Media Services SDK without using Media Services APIs.

Next steps

To learn how to attach a storage account to your Media Services account, see [Create an account](#).

Handling Event Grid events

10/17/2018 • 2 minutes to read • [Edit Online](#)

Media Services events allow applications to react to different events (for example, the job state change event) using modern serverless architectures. It does so without the need for complicated code or expensive and inefficient polling services. Instead, events are pushed through [Azure Event Grid](#) to event handlers such as [Azure Functions](#), [Azure Logic Apps](#), or even to your own Webhook, and you only pay for what you use. For information about pricing, see [Event Grid pricing](#).

Availability for Media Services events is tied to Event Grid [availability](#) and will become available in other regions as Event Grid does.

Available Media Services events

Event grid uses [event subscriptions](#) to route event messages to subscribers. Currently, Media Services event subscriptions can include the following events:

EVENT NAME	DESCRIPTION
Microsoft.Media.JobStateChange	Raised when a state of the job changes.
Microsoft.Media.LiveEventConnectionRejected	Encoder's connection attempt is rejected.
Microsoft.Media.LiveEventEncoderConnected	Encoder establishes connection with live event.
Microsoft.Media.LiveEventEncoderDisconnected	Encoder disconnects.
Microsoft.Media.LiveEventIncomingDataChunkDropped	Media server drops data chunk because it's too late or has an overlapping timestamp (timestamp of new data chunk is less than the end time of the previous data chunk).
Microsoft.Media.LiveEventIncomingStreamReceived	Media server receives first data chunk for each track in the stream or connection.
Microsoft.Media.LiveEventIncomingStreamsOutOfSync	Media server detects audio and video streams are out of sync. Use as a warning because user experience may not be impacted.
Microsoft.Media.LiveEventIncomingVideoStreamsOutOfSync	Media server detects any of the two video streams coming from external encoder are out of sync. Use as a warning because user experience may not be impacted.
Microsoft.Media.LiveEventIngestHeartbeat	Published every 20 seconds for each track when live event is running. Provides ingest health summary.
Microsoft.Media.LiveEventTrackDiscontinuityDetected	Media server detects discontinuity in the incoming track.

Event Schema

Media Services events contain all the information you need to respond to changes in your data. You can identify a Media Services event because the `eventType` property starts with "Microsoft.Media.".

For more information, see [Media Services event schemas](#).

Practices for consuming events

Applications that handle Media Services events should follow a few recommended practices:

- As multiple subscriptions can be configured to route events to the same event handler, it is important not to assume events are from a particular source, but to check the topic of the message to ensure that it comes from the storage account you are expecting.
- Similarly, check that the `eventType` is one you are prepared to process, and do not assume that all events you receive will be the types you expect.
- Ignore fields you don't understand. This practice will help keep you resilient to new features that might be added in the future.
- Use the "subject" prefix and suffix matches to limit events to a particular event.

Next steps

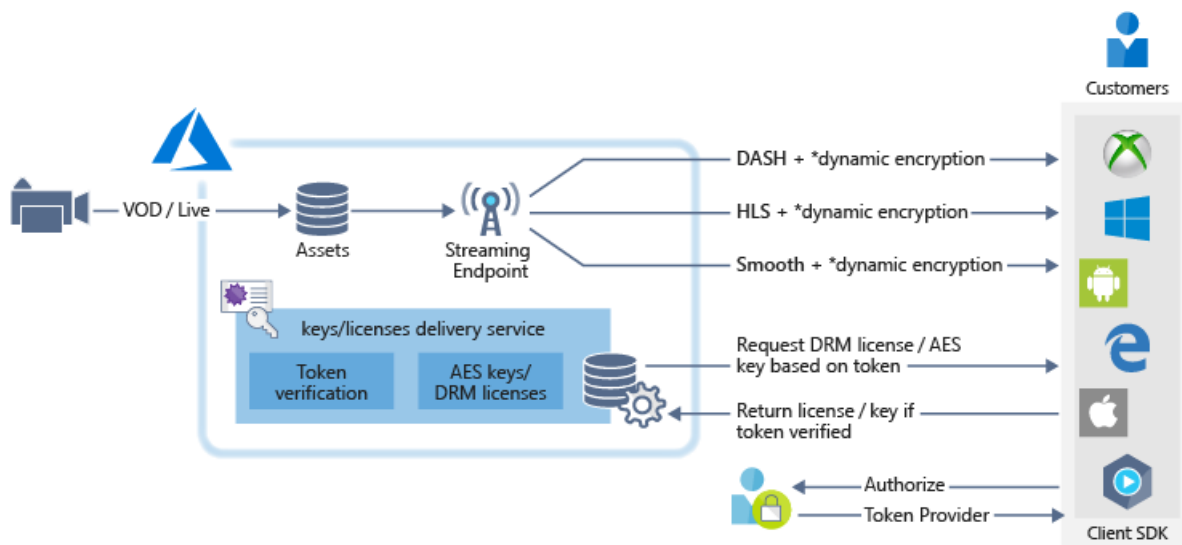
[Get job state events](#)

Content protection overview

12/10/2018 • 9 minutes to read • [Edit Online](#)

You can use Azure Media Services to secure your media from the time it leaves your computer through storage, processing, and delivery. With Media Services, you can deliver your live and on-demand content encrypted dynamically with Advanced Encryption Standard (AES-128) or any of the three major digital rights management (DRM) systems: Microsoft PlayReady, Google Widevine, and Apple FairPlay. Media Services also provides a service for delivering AES keys and DRM (PlayReady, Widevine, and FairPlay) licenses to authorized clients.

The following image illustrates the Media Services content protection workflow:



* *dynamic encryption supports AES-128 "clear key", CBCS, and CENC. For details see the support matrix [here](#).*

This article explains concepts and terminology relevant to understanding content protection with Media Services. The article also has the [FAQ](#) section and provides links to articles that show how to protect content.

Main components of the content protection system

To successfully complete your "content protection" system/application design, you need to fully understanding the scope of the effort. The following list gives an overview of three parts you would need to implement.

1. Azure Media Services code

- License templates for PlayReady, Widevine and/or FairPlay. The templates let you configure rights and permissions for each of the used DRMs
- License delivery authorization, specifying the logic of authorization check based on claims in JWT
- Content keys, streaming protocols and corresponding DRMs applied, defining DRM encryption

NOTE

You can encrypt each asset with multiple encryption types (AES-128, PlayReady, Widevine, FairPlay). See [Streaming protocols and encryption types](#), to see what makes sense to combine.

The following articles show steps for encrypting content with AES and/or DRM:

- [Protect with AES encryption](#)
- [Protect with DRM](#)

2. Player with AES or DRM client. A video player app based on a player SDK (either native or browser-based) needs to meet the following requirements:

- The player SDK supports the needed DRM clients
- The player SDK supports the required streaming protocols: Smooth, DASH, and/or HLS
- The player SDK needs to be able to handle passing a JWT token in license acquisition request

You can create a player by using the [Azure Media Player API](#). Use the [Azure Media Player ProtectionInfo API](#) to specify which DRM technology to use on different DRM platforms.

For testing AES or CENC (Widevine and/or PlayReady) encrypted content, you can use [Azure Media Player](#). Make sure you click on "Advanced options" and check your encryption options.

If you want to test FairPlay encrypted content, use [this test player](#). The player supports Widevine, PlayReady, and FairPlay DRMs as well as AES-128 clear key encryption. You need to choose the right browser to test different DRMs: Chrome/Opera/Firefox for Widevine, MS Edge/IE11 for PlayReady, Safari on macOS for FairPlay.

3. Secure Token Service (STS), which issues JSON Web Token (JWT) as access token for backend resource access. You can use the AMS license delivery services as the backend resource. An STS has to define the following:

- Issuer and audience (or scope)
- Claims, which are dependent on business requirements in content protection
- Symmetric or asymmetric verification for signature verification
- Key rollover support (if necessary)

You can use [this STS tool](#) to test STS, which supports all 3 types of verification key: symmetric, asymmetric, or AAD with key rollover.

NOTE

It is highly recommended to focus and fully test each part (described above) before moving onto the next part. To test your "content protection" system, use the tools specified in the list above.

Streaming protocols and encryption types

You can use Media Services to deliver your content encrypted dynamically with AES clear key or DRM encryption by using PlayReady, Widevine, or FairPlay. Currently, you can encrypt the HTTP Live Streaming (HLS), MPEG DASH, and Smooth Streaming formats. Each protocol supports the following encryption methods:

PROTOCOL	CONTAINER FORMAT	ENCRYPTION SCHEME
MPEG-DASH	All	AES
	CSF(fmp4)	CENC (Widevine + PlayReady)
	CMAF(fmp4)	CENC (Widevine + PlayReady)
HLS	All	AES
	MPG2-TS	CBCS (Fairplay)
	MPG2-TS	CENC (PlayReady)

PROTOCOL	CONTAINER FORMAT	ENCRYPTION SCHEME
	CMAF(fmp4)	CENC (PlayReady)
Smooth Streaming	fMP4	AES
	fMP4	CENC (PlayReady)

Dynamic encryption

In Media Services v3, a content key is associated with StreamingLocator (see [this example](#)). If using the Media Services key delivery service, you should auto generate the content key. You should generate the content key yourself if you are using your own key delivery service, or if you need to handle a high availability scenario where you need to have the same content key in two datacenters.

When a stream is requested by a player, Media Services uses the specified key to dynamically encrypt your content by using AES clear key or DRM encryption. To decrypt the stream, the player requests the key from Media Services key delivery service or the key delivery service you specified. To decide whether or not the user is authorized to get the key, the service evaluates the content key policy that you specified for the key.

AES-128 clear key vs. DRM

Customers often wonder whether they should use AES encryption or a DRM system. The primary difference between the two systems is that with AES encryption the content key is transmitted to the client in an unencrypted format ("in the clear"). As a result, the key used to encrypt the content can be viewed in a network trace on the client in plain text. AES-128 clear key encryption is suitable for use cases where the viewer is a trusted party (for example, encrypting corporate videos distributed within a company to be viewed by employees).

PlayReady, Widevine, and FairPlay all provide a higher level of encryption compared to AES-128 clear key encryption. The content key is transmitted in an encrypted format. Additionally, decryption is handled in a secure environment at the operating system level, where it's more difficult for a malicious user to attack. DRM is recommended for use cases where the viewer might not be a trusted party and you require the highest level of security.

Storage side encryption

To protect your Assets at rest, the assets should be encrypted by the storage side encryption. The following table shows how the storage side encryption works in Media Services v3:

ENCRYPTION OPTION	DESCRIPTION	MEDIA SERVICES V3
Media Services Storage Encryption	AES-256 encryption, key managed by Media Services	Not supported ⁽¹⁾
Storage Service Encryption for Data at Rest	Server-side encryption offered by Azure Storage, key managed by Azure or by customer	Supported
Storage Client-Side Encryption	Client-side encryption offered by Azure storage, key managed by customer in Key Vault	Not supported

¹ In Media Services v3, storage encryption (AES-256 encryption) is only supported for backwards compatibility when your Assets were created with Media Services v2. Meaning v3 works with existing storage encrypted assets

but will not allow creation of new ones.

Licenses and keys delivery service

Media Services provides a key delivery service for delivering DRM (PlayReady, Widevine, FairPlay) licenses and AES keys to authorized clients. You can use the REST API, or a Media Services client library to configure authorization and authentication policies for your licenses and keys.

Control content access

You can control who has access to your content by configuring the content key policy. Media Services supports multiple ways of authenticating users who make key requests. You must configure the content key policy. The client (player) must meet the policy before the key can be delivered to the client. The content key policy can have **open** or **token** restriction.

With a token-restricted content key policy, the content key is sent only to a client that presents a valid JSON Web Token (JWT) or simple web token (SWT) in the key/license request. This token must be issued by a security token service (STS). You can use Azure Active Directory as an STS or deploy a custom STS. The STS must be configured to create a token signed with the specified key and issue claims that you specified in the token restriction configuration. The Media Services key delivery service returns the requested key/license to the client if the token is valid and the claims in the token match those configured for the key/license.

When you configure the token restricted policy, you must specify the primary verification key, issuer, and audience parameters. The primary verification key contains the key that the token was signed with. The issuer is the secure token service that issues the token. The audience, sometimes called scope, describes the intent of the token or the resource the token authorizes access to. The Media Services key delivery service validates that these values in the token match the values in the template.

Frequently asked questions

Question

How to implement multi-DRM (PlayReady, Widevine, and FairPlay) system using Azure Media Services (AMS) v3 and also use AMS license/key delivery service?

Answer

For end-to-end scenario, see the [following code example](#).

The example shows how to:

1. Create and configure ContentKeyPolicies.

The sample contains functions that configure [PlayReady](#), [Widevine](#), and [FairPlay](#) licenses.

```
ContentKeyPolicyPlayReadyConfiguration playReadyConfig = ConfigurePlayReadyLicenseTemplate();
ContentKeyPolicyWidevineConfiguration widevineConfig = ConfigureWidevineLicenseTemplate();
ContentKeyPolicyFairPlayConfiguration fairPlayConfig = ConfigureFairPlayPolicyOptions();
```

2. Create a StreamingLocator that is configured to stream an encrypted asset.

For example, you can set StreamingLocator.StreamingPolicyName to the "Predefined_MultiDrmCencStreaming" policy. This policy indicates that you want for two content keys (envelope and CENC) to get generated and set on the locator. Thus, the envelope, PlayReady, and Widevine encryptions are applied (the key is delivered to the playback client based on the configured DRM licenses). If you also want to encrypt your stream with CBCS (FairPlay), use "Predefined_MultiDrmStreaming".

3. Create a test token.

The **GetTokenAsync** method shows how to create a test token.

4. Build the streaming URL.

The **GetDASHStreamingUrlAsync** method shows how to build the streaming URL. In this case, the URL streams the **DASH** content.

Question

How and where to get JWT token before using it to request license or key?

Answer

1. For production, you need to have a Secure Token Services (STS) (web service) which issues JWT token upon a HTTPS request. For test, you could use the code shown in **GetTokenAsync** method defined in [Program.cs](#).
 2. Player will need to make a request, after a user is authenticated, to the STS for such a token and assign it as the value of the token. You can use the [Azure Media Player API](#).
- For an example of running STS, with either symmetric and asymmetric key, please see <http://aka.ms/jwt>.
 - For an example of a player based on Azure Media Player using such JWT token, see <http://aka.ms/amtest> (expand "player_settings" link to see the token input).

Question

How do you authorize requests to stream videos with AES encryption?

Answer

The correct approach is to leverage STS (Secure Token Service):

In STS, depending on user profile, add different claims (such as "Premium User", "Basic User", "Free Trial User"). With different claims in a JWT, the user can see different contents. Of course, for different content/asset, the ContentKeyPolicyRestriction will have the corresponding RequiredClaims.

Use Azure Media Services APIs for configuring license/key delivery and encrypting your assets (as shown in [this sample](#)).

Next steps

Check out the following articles:

- [Protect with AES encryption](#)
- [Protect with DRM](#)

Additional information can be found in [Design multi-drm content protection system with access control](#)

Design of a multi-DRM content protection system with access control

12/10/2018 • 24 minutes to read • [Edit Online](#)

Overview

Designing and building a Digital Rights Management (DRM) subsystem for an over-the-top (OTT) or online streaming solution is a complex task. Operators/online video providers typically outsource this task to specialized DRM service providers. The goal of this document is to present a reference design and a reference implementation of an end-to-end DRM subsystem in an OTT or online streaming solution.

The targeted readers for this document are engineers who work in DRM subsystems of OTT or online streaming/multiscreen solutions or readers who are interested in DRM subsystems. The assumption is that readers are familiar with at least one of the DRM technologies on the market, such as PlayReady, Widevine, FairPlay, or Adobe Access.

In this discussion, by multi-DRM we include the 3 DRMs supported by Azure Media Services: Common Encryption (CENC) for PlayReady and Widevine, FairPlay as well as AES-128 clear key encryption. A major trend in online streaming and the OTT industry is to use native DRMs on various client platforms. This trend is a shift from the previous one that used a single DRM and its client SDK for various client platforms. When you use CENC with multi-native DRM, both PlayReady and Widevine are encrypted per the [Common Encryption \(ISO/IEC 23001-7 CENC\)](#) specification.

The benefits of using native multi-DRM for content protection are that it:

- Reduces encryption cost because a single encryption process is used to target different platforms with its native DRMs.
- Reduces the cost of managing assets because only a single copy of asset is needed in storage.
- Eliminates DRM client licensing cost because the native DRM client is usually free on its native platform.

Goals of the article

The goals of this article are to:

- Provide a reference design of a DRM subsystem that uses all 3 DRMs (CENC for DASH, FairPlay for HLS and PlayReady for smooth streaming).
- Provide a reference implementation on Azure and Azure Media Services platform.
- Discuss some design and implementation topics.

The following table summarizes native DRM support on different platforms and EME support in different browsers.

CLIENT PLATFORM	NATIVE DRM	EME
Smart TVs, STBs	PlayReady, Widevine, and/or other	Embedded browser/EME for PlayReady and/or Widevine
Windows 10	PlayReady	MS Edge/IE11 for PlayReady
Android devices (phone, tablet, TV)	Widevine	Chrome for Widevine

CLIENT PLATFORM	NATIVE DRM	EME
iOS	FairPlay	Safari for FairPlay (since iOS 11.2)
macOS	FairPlay	Safari for FairPlay (since Safari 9+ on Mac OS X 10.11+ El Capitan)
tvOS	FairPlay	

Considering the current state of deployment for each DRM, a service typically wants to implement two or three DRMs to make sure you address all the types of endpoints in the best way.

There is a tradeoff between the complexity of the service logic and the complexity on the client side to reach a certain level of user experience on the various clients.

To make your selection, keep in mind:

- PlayReady is natively implemented in every Windows device, on some Android devices, and available through software SDKs on virtually any platform.
- Widevine is natively implemented in every Android device, in Chrome, and in some other devices. Widevine is also supported in Firefox and Opera browsers over DASH.
- FairPlay is available on iOS, macOS and tvOS.

A reference design

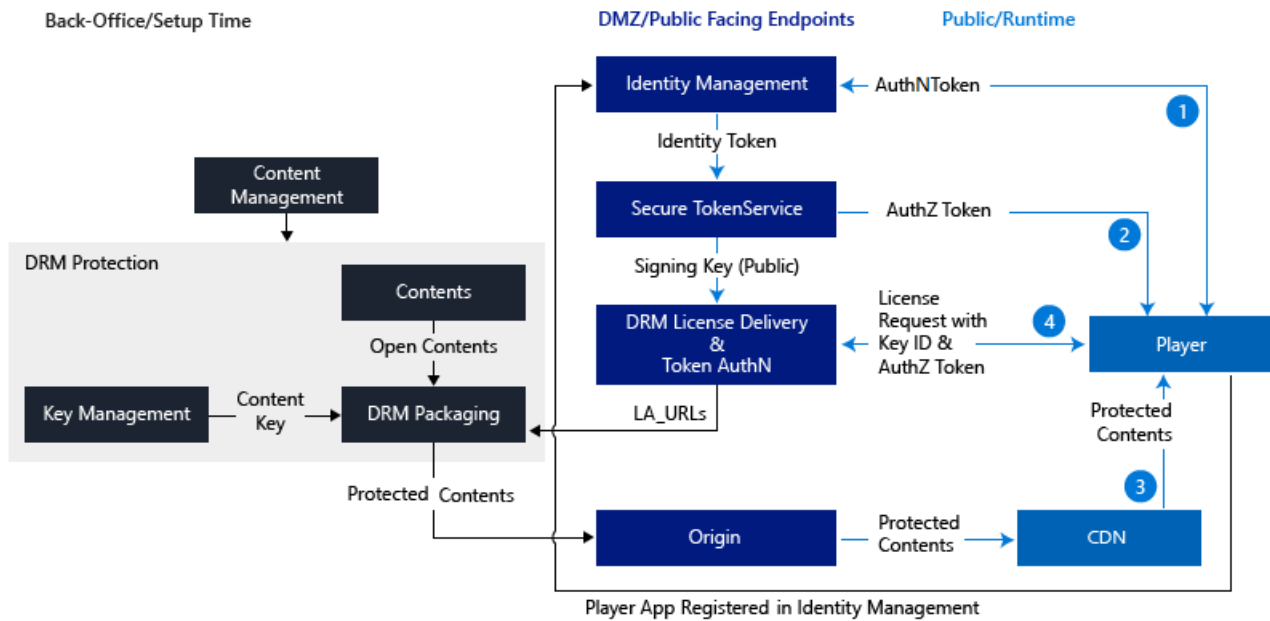
This section presents a reference design that is agnostic to the technologies used to implement it.

A DRM subsystem can contain the following components:

- Key management
- DRM encryption packaging
- DRM license delivery
- Entitlement check/access control
- User authentication/authorization
- Player app
- Origin/content delivery network (CDN)

The following diagram illustrates the high-level interaction among the components in a DRM subsystem:

Logical Diagram of a Generic DRM Subsystem with CENC



The design has three basic layers:

- A back-office layer (black) is not exposed externally.
- A DMZ layer (dark blue) contains all the endpoints that face the public.
- A public internet layer (light blue) contains the CDN and players with traffic across the public internet.

There also should be a content management tool to control DRM protection, regardless of whether it's static or dynamic encryption. The inputs for DRM encryption include:

- MBR video content
- Content key
- License acquisition URLs

Here's the high-level flow during playback time:

- The user is authenticated.
- An authorization token is created for the user.
- DRM protected content (manifest) is downloaded to the player.
- The player submits a license acquisition request to license servers together with a key ID and an authorization token.

The following section discusses the design of key management.

CONTENTKEY-TO-ASSET	SCENARIO
1-to-1	The simplest case. It provides the finest control. But this arrangement generally results in the highest license delivery cost. At minimum, one license request is required for each protected asset.
1-to-many	You could use the same content key for multiple assets. For example, for all the assets in a logical group, such as a genre or the subset of a genre (or movie gene), you can use a single content key.

CONTENTKEY-TO-ASSET	SCENARIO
Many-to-1	<p>Multiple content keys are needed for each asset.</p> <p>For example, if you need to apply dynamic CENC protection with multi-DRM for MPEG-DASH and dynamic AES-128 encryption for HLS, you need two separate content keys. Each content key needs its own ContentKeyType. (For the content key used for dynamic CENC protection, use ContentKeyType.CommonEncryption. For the content key used for dynamic AES-128 encryption, use ContentKeyType.EnvelopeEncryption.)</p> <p>As another example, in CENC protection of DASH content, in theory, you can use one content key to protect the video stream and another content key to protect the audio stream.</p>
Many-to-many	Combination of the previous two scenarios. One set of content keys is used for each of the multiple assets in the same asset group.

Another important factor to consider is the use of persistent and nonpersistent licenses.

Why are these considerations important?

If you use a public cloud for license delivery, persistent and nonpersistent licenses have a direct impact on license delivery cost. The following two different design cases serve to illustrate:

- Monthly subscription: Use a persistent license and 1-to-many content key-to-asset mapping. For example, for all the kids' movies, we use a single content key for encryption. In this case:

Total number of licenses requested for all kids' movies/device = 1

- Monthly subscription: Use a nonpersistent license and 1-to-1 mapping between content key and asset. In this case:

Total number of licenses requested for all kids' movies/device = [number of movies watched] x [number of sessions]

The two different designs result in very different license request patterns. The different patterns result in different license delivery cost if license delivery service is provided by a public cloud such as Media Services.

Map design to technology for implementation

Next, the generic design is mapped to technologies on the Azure/Media Services platform by specifying which technology to use for each building block.

The following table shows the mapping.

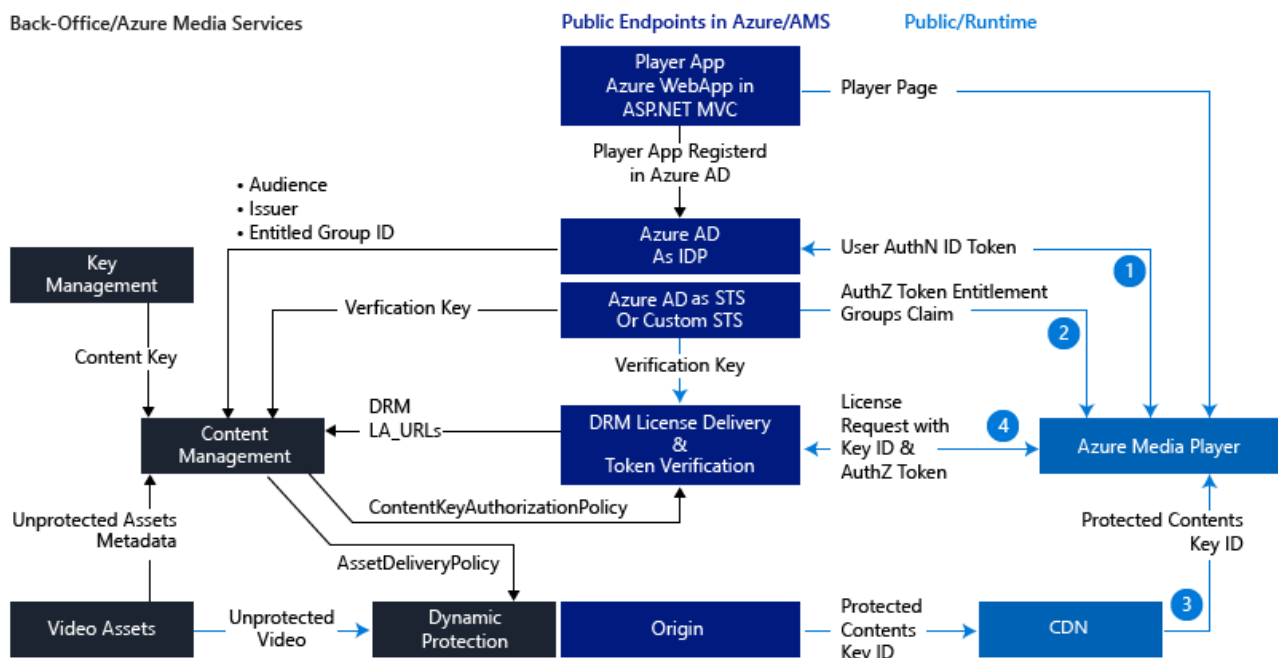
BUILDING BLOCK	TECHNOLOGY
Player	Azure Media Player
Identity Provider (IDP)	Azure Active Directory (Azure AD)
Secure Token Service (STS)	Azure AD
DRM protection workflow	Azure Media Services dynamic protection

BUILDING BLOCK	TECHNOLOGY
DRM license delivery	<ul style="list-style-type: none"> * Media Services license delivery (PlayReady, Widevine, FairPlay) * Axinom license server * Custom PlayReady license server
Origin	Azure Media Services streaming endpoint
Key management	Not needed for reference implementation
Content management	A C# console application

In other words, both IDP and STS are provided by Azure AD. The [Azure Media Player API](#) is used for the player. Both Azure Media Services and Azure Media Player support CENC over DASH, FairPlay over HLS, PlayReady over smooth streaming, and AES-128 encryption for DASH, HLS and smooth.

The following diagram shows the overall structure and flow with the previous technology mapping:

CENC Subsystem Reference Implementation on Azure/Azure Media Services Platform



To set up DRM content protection, the content management tool uses the following inputs:

- Open content
- Content key from key management
- License acquisition URLs
- A list of information from Azure AD, such as audience, issuer, and token claims

Here's the output of the content management tool:

- ContentKeyPolicy describes DRM license template for each kind of DRM used;
- ContentKeyPolicyRestriction describes the access control before a DRM license is issued
- Streamingpolicy describes the various combinations of DRM - encryption mode - streaming protocol - container format, for streaming
- StreamingLocator describes content key/IV used for encryption, and streaming URLs

Here's the flow during runtime:

- Upon user authentication, a JWT is generated.
- One of the claims contained in the JWT is a groups claim that contains the group object ID EntitledUserGroup. This claim is used to pass the entitlement check.
- The player downloads the client manifest of CENC-protected content and identifies the following:
 - Key ID.
 - The content is DRM protected.
 - License acquisition URLs.
- The player makes a license acquisition request based on the browser/DRM supported. In the license acquisition request, the key ID and the JWT are also submitted. The license delivery service verifies the JWT and the claims contained before it issues the needed license.

Implementation

Implementation procedures

Implementation includes the following steps:

1. Prepare test assets. Encode/package a test video to multi-bitrate fragmented MP4 in Media Services. This asset is *not* DRM protected. DRM protection is done by dynamic protection later.
2. Create a key ID and a content key (optionally from a key seed). In this instance, the key management system isn't needed because only a single key ID and content key are required for a couple of test assets.
3. Use the Media Services API to configure multi-DRM license delivery services for the test asset. If you use custom license servers by your company or your company's vendors instead of license services in Media Services, you can skip this step. You can specify license acquisition URLs in the step when you configure license delivery. The Media Services API is needed to specify some detailed configurations, such as authorization policy restriction and license response templates for different DRM license services. At this time, the Azure portal doesn't provide the needed UI for this configuration. For API-level information and sample code, see [Use PlayReady and/or Widevine dynamic common encryption](#).
4. Use the Media Services API to configure the asset delivery policy for the test asset. For API-level information and sample code, see [Use PlayReady and/or Widevine dynamic common encryption](#).
5. Create and configure an Azure AD tenant in Azure.
6. Create a few user accounts and groups in your Azure AD tenant. Create at least an "Entitled User" group, and add a user to this group. Users in this group pass the entitlement check in license acquisition. Users not in this group fail to pass the authentication check and can't acquire a license. Membership in this "Entitled User" group is a required groups claim in the JWT issued by Azure AD. You specify this claim requirement in the step when you configure multi-DRM license delivery services.
7. Create an ASP.NET MVC app to host your video player. This ASP.NET app is protected with user authentication against the Azure AD tenant. Proper claims are included in the access tokens obtained after user authentication. We recommend OpenID Connect API for this step. Install the following NuGet packages:
 - Install-Package Microsoft.Azure.ActiveDirectory.GraphClient
 - Install-Package Microsoft.Owin.Security.OpenIdConnect
 - Install-Package Microsoft.Owin.Security.Cookies
 - Install-Package Microsoft.Owin.Host.SystemWeb
 - Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
8. Create a player by using the [Azure Media Player API](#). Use the [Azure Media Player ProtectionInfo API](#) to specify which DRM technology to use on different DRM platforms.
9. The following table shows the test matrix.

DRM	BROWSER	RESULT FOR ENTITLED USER	RESULT FOR UNENTITLED USER
PlayReady	Microsoft Edge or Internet Explorer 11 on Windows 10	Succeed	Fail
Widevine	Chrome, Firefox, Opera	Succeed	Fail
FairPlay	Safari on macOS	Succeed	Fail
AES-128	Most modern browsers	Succeed	Fail

For information on how to set up Azure AD for an ASP.NET MVC player app, see [Integrate an Azure Media Services OWIN MVC-based app with Azure Active Directory and restrict content key delivery based on JWT claims](#).

For more information, see [JWT token authentication in Azure Media Services and dynamic encryption](#).

For information on Azure AD:

- You can find developer information in the [Azure Active Directory developer's guide](#).
- You can find administrator information in [Administer your Azure AD tenant directory](#).

Some issues in implementation

Use the following troubleshooting information for help with implementation issues.

- The issuer URL must end with "/". The audience must be the player application client ID. Also, add "/" at the end of the issuer URL.

```
<add key="ida:audience" value="[Application Client ID GUID]" />
<add key="ida:issuer" value="https://sts.windows.net/[AAD Tenant ID]/" />
```

In the [JWT Decoder](#), you see **aud** and **iss**, as shown in the JWT:

```
aud: "ec...f90c",
iss: "https://sts.windows.net/51e...85/",
iat: 1448556434,
nbf: 1448556434,
exp: 1448560334,
```

- Add permissions to the application in Azure AD on the **Configure** tab of the application. Permissions are required for each application, both local and deployed versions.

permissions to other applications

- Use the correct issuer when you set up dynamic CENC protection.

```
<add key="ida:issuer" value="https://sts.windows.net/[AAD Tenant ID]/"/>
```


The following doesn't work:

```
<add key="ida:issuer" value="https://willzhanad.onmicrosoft.com/" />
```

The GUID is the Azure AD tenant ID. The GUID can be found in the **Endpoints** pop-up menu in the Azure portal.

- Grant group membership claims privileges. Make sure the following is in the Azure AD application manifest file:

"groupMembershipClaims": "All" (the default value is null)

- Set the proper TokenType when you create restriction requirements.

```
objTokenRestrictionTemplate.TokenType = TokenType.JWT;
```

Because you add support for JWT (Azure AD) in addition to SWT (ACS), the default TokenType is TokenType.JWT. If you use SWT/ACS, you must set the token to TokenType.SWT.

FAQs

This section discusses some additional topics in design and implementation.

HTTP or HTTPS?

The ASP.NET MVC player application must support the following:

- User authentication through Azure AD, which is under HTTPS.
- JWT exchange between the client and Azure AD, which is under HTTPS.
- DRM license acquisition by the client, which must be under HTTPS if license delivery is provided by Media Services. The PlayReady product suite doesn't mandate HTTPS for license delivery. If your PlayReady license server is outside Media Services, you can use either HTTP or HTTPS.

The ASP.NET player application uses HTTPS as a best practice, so Media Player is on a page under HTTPS. However, HTTP is preferred for streaming, so you need to consider the issue of mixed content.

- The browser doesn't allow mixed content. But plug-ins like Silverlight and the OSMF plug-in for smooth and DASH do allow it. Mixed content is a security concern because of the threat of the ability to inject malicious JavaScript, which can cause customer data to be at risk. Browsers block this capability by default. The only way to work around it is on the server (origin) side by allowing all domains (regardless of HTTPS or HTTP). This is probably not a good idea either.
- Avoid mixed content. Both the player application and Media Player should use HTTP or HTTPS. When playing mixed content, the silverlightSS tech requires clearing a mixed-content warning. The flashSS tech handles mixed content without a mixed-content warning.
- If your streaming endpoint was created before August 2014, it won't support HTTPS. In this case, create and use a new streaming endpoint for HTTPS.

In the reference implementation for DRM-protected contents, both the application and streaming are under HTTPS. For open contents, the player doesn't need authentication or a license, so you can use either HTTP or HTTPS.

What is Azure Active Directory signing key rollover?

Signing key rollover is an important point to take into consideration in your implementation. If you ignore it, the finished system eventually stops working completely, within six weeks at the most.

Azure AD uses industry standards to establish trust between itself and applications that use Azure AD. Specifically,

Azure AD uses a signing key that consists of a public and private key pair. When Azure AD creates a security token that contains information about the user, it's signed by Azure AD with a private key before it's sent back to the application. To verify that the token is valid and originated from Azure AD, the application must validate the token's signature. The application uses the public key exposed by Azure AD that is contained in the tenant's federation metadata document. This public key, and the signing key from which it derives, is the same one used for all tenants in Azure AD.

For more information on Azure AD key rollover, see [Important information about signing key rollover in Azure AD](#).

Between the [public-private key pair](#):

- The private key is used by Azure AD to generate a JWT.
- The public key is used by an application such as DRM license delivery services in Media Services to verify the JWT.

For security purposes, Azure AD rotates the certificate periodically (every six weeks). In the case of security breaches, the key rollover can occur any time. Therefore, the license delivery services in Media Services need to update the public key used as Azure AD rotates the key pair. Otherwise, token authentication in Media Services fails and no license is issued.

To set up this service, you set `TokenRestrictionTemplate.OpenIdConnectDiscoveryDocument` when you configure DRM license delivery services.

Here's the JWT flow:

- Azure AD issues the JWT with the current private key for an authenticated user.
- When a player sees a CENC with multi-DRM protected content, it first locates the JWT issued by Azure AD.
- The player sends a license acquisition request with the JWT to license delivery services in Media Services.
- The license delivery services in Media Services use the current/valid public key from Azure AD to verify the JWT before issuing licenses.

DRM license delivery services always check for the current/valid public key from Azure AD. The public key presented by Azure AD is the key used to verify a JWT issued by Azure AD.

What if the key rollover happens after Azure AD generates a JWT but before the JWT is sent by players to DRM license delivery services in Media Services for verification?

Because a key can be rolled over at any moment, more than one valid public key is always available in the federation metadata document. Media Services license delivery can use any of the keys specified in the document. Because one key might be rolled soon, another might be its replacement, and so forth.

Where is the access token?

If you look at how a web app calls an API app under [Application identity with OAuth 2.0 client credentials grant](#), the authentication flow is as follows:

- A user signs in to Azure AD in the web application. For more information, see [Web browser to web application](#).
- The Azure AD authorization endpoint redirects the user agent back to the client application with an authorization code. The user agent returns the authorization code to the client application's redirect URI.
- The web application needs to acquire an access token so that it can authenticate to the web API and retrieve the desired resource. It makes a request to the Azure AD token endpoint and provides the credential, client ID, and web API's application ID URI. It presents the authorization code to prove that the user consented.
- Azure AD authenticates the application and returns a JWT access token that's used to call the web API.
- Over HTTPS, the web application uses the returned JWT access token to add the JWT string with a "Bearer" designation in the "Authorization" header of the request to the web API. The web API then validates the JWT. If validation is successful, it returns the desired resource.

In this application identity flow, the web API trusts that the web application authenticated the user. For this reason, this pattern is called a trusted subsystem. The [authorization flow diagram](#) describes how authorization-code-grant flow works.

License acquisition with token restriction follows the same trusted subsystem pattern. The license delivery service in Media Services is the web API resource, or the "back-end resource" that a web application needs to access. So where is the access token?

The access token is obtained from Azure AD. After successful user authentication, an authorization code is returned. The authorization code is then used, together with the client ID and the app key, to exchange for the access token. The access token is used to access a "pointer" application that points to or represents the Media Services license delivery service.

To register and configure the pointer app in Azure AD, take the following steps:

1. In the Azure AD tenant:
 - Add an application (resource) with the sign-on URL `https://[resource_name].azurewebsites.net/`.
 - Add an app ID with the URL `https://[aad_tenant_name].onmicrosoft.com/[resource_name]`.
2. Add a new key for the resource app.
3. Update the app manifest file so that the `groupMembershipClaims` property has the value `"groupMembershipClaims": "All"`.
4. In the Azure AD app that points to the player web app, in the section **Permissions to other applications**, add the resource app that was added in step 1. Under **Delegated permission**, select **Access [resource_name]**. This option gives the web app permission to create access tokens that access the resource app. Do this for both the local and deployed version of the web app if you develop with Visual Studio and the Azure web app.

The JWT issued by Azure AD is the access token used to access the pointer resource.

What about live streaming?

The previous discussion focused on on-demand assets. What about live streaming?

You can use exactly the same design and implementation to protect live streaming in Media Services by treating the asset associated with a program as a VOD asset.

Specifically, to do live streaming in Media Services, you need to create a channel and then create a program under the channel. To create the program, you need to create an asset that contains the live archive for the program. To provide CENC with multi-DRM protection of the live content, apply the same setup/processing to the asset as if it were a VOD asset before you start the program.

What about license servers outside Media Services?

Often, customers invested in a license server farm either in their own data center or one hosted by DRM service providers. With Media Services content protection, you can operate in hybrid mode. Contents can be hosted and dynamically protected in Media Services, while DRM licenses are delivered by servers outside Media Services. In this case, consider the following changes:

- STS needs to issue tokens that are acceptable and can be verified by the license server farm. For example, the Widevine license servers provided by Axinom require a specific JWT that contains an entitlement message. Therefore, you need to have an STS to issue such a JWT.
- You no longer need to configure license delivery service in Media Services. You need to provide the license acquisition URLs (for PlayReady, Widevine, and FairPlay) when you configure `ContentKeyPolicies`.

What if I want to use a custom STS?

A customer might choose to use a custom STS to provide JWTs. Reasons include:

- The IDP used by the customer doesn't support STS. In this case, a custom STS might be an option.
- The customer might need more flexible or tighter control to integrate STS with the customer's subscriber billing system. For example, an MVPD operator might offer multiple OTT subscriber packages, such as premium, basic, and sports. The operator might want to match the claims in a token with a subscriber's package so that only the contents in a specific package are made available. In this case, a custom STS provides the needed flexibility and control.

When you use a custom STS, two changes must be made:

- When you configure license delivery service for an asset, you need to specify the security key used for verification by the custom STS instead of the current key from Azure AD. (More details follow.)
- When a JWT token is generated, a security key is specified instead of the private key of the current X509 certificate in Azure AD.

There are two types of security keys:

- Symmetric key: The same key is used to generate and to verify a JWT.
- Asymmetric key: A public-private key pair in an X509 certificate is used with a private key to encrypt/generate a JWT and with the public key to verify the token.

NOTE

If you use .NET Framework/C# as your development platform, the X509 certificate used for an asymmetric security key must have a key length of at least 2048. This is a requirement of the class `System.IdentityModel.Tokens.X509AsymmetricSecurityKey` in .NET Framework. Otherwise, the following exception is thrown:

IDX10630: The 'System.IdentityModel.Tokens.X509AsymmetricSecurityKey' for signing cannot be smaller than '2048' bits.

The completed system and test

This section walks you through the following scenarios in the completed end-to-end system so that you can have a basic picture of the behavior before you get a sign-in account:

- If you need a non-integrated scenario:
 - For video assets hosted in Media Services that are either unprotected or DRM protected but without token authentication (issuing a license to whoever requested it), you can test it without signing in. Switch to HTTP if your video streaming is over HTTP.
- If you need an end-to-end integrated scenario:
 - For video assets under dynamic DRM protection in Media Services, with the token authentication and JWT generated by Azure AD, you need to sign in.

For the player web application and its sign-in, see [this website](#).

User sign-in

To test the end-to-end integrated DRM system, you need to have an account created or added.

What account?

Although Azure originally allowed access only by Microsoft account users, access is now allowed by users from both systems. All Azure properties now trust Azure AD for authentication, and Azure AD authenticates organizational users. A federation relationship was created where Azure AD trusts the Microsoft account consumer identity system to authenticate consumer users. As a result, Azure AD can authenticate guest Microsoft accounts as well as native Azure AD accounts.

Because Azure AD trusts the Microsoft account domain, you can add any accounts from any of the following

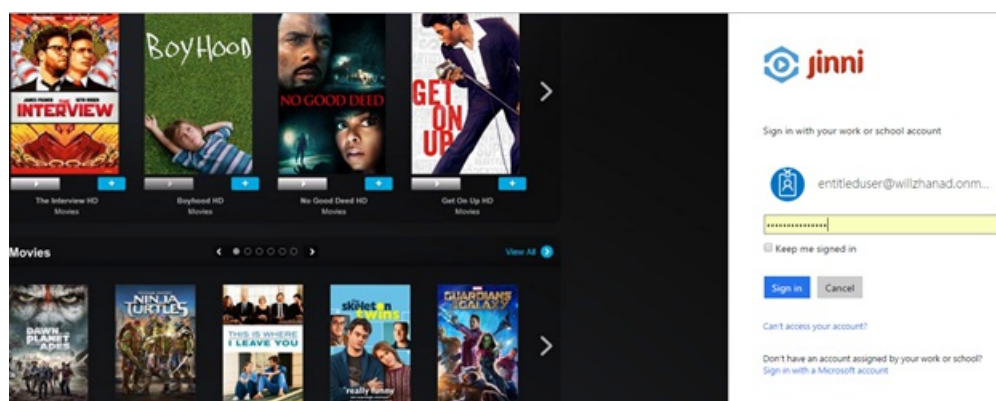
domains to the custom Azure AD tenant and use the account to sign in:

DOMAIN NAME	DOMAIN
Custom Azure AD tenant domain	somename.onmicrosoft.com
Corporate domain	microsoft.com
Microsoft account domain	outlook.com, live.com, hotmail.com

You can contact any of the authors to have an account created or added for you.

The following screenshots show different sign-in pages used by different domain accounts:

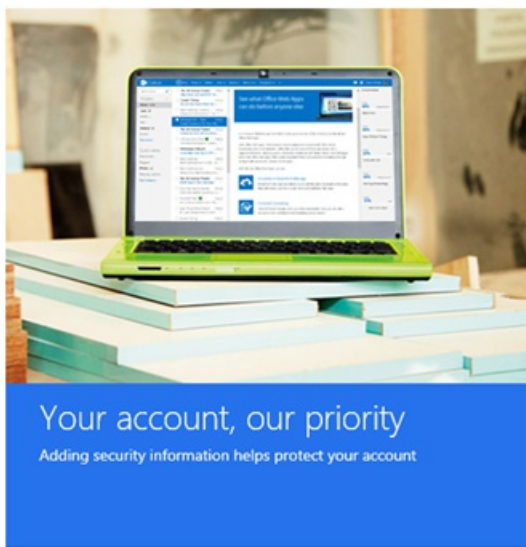
Custom Azure AD tenant domain account: The customized sign-in page of the custom Azure AD tenant domain.



Microsoft domain account with smart card: The sign-in page customized by Microsoft corporate IT with two-factor authentication.



Microsoft account: The sign-in page of the Microsoft account for consumers.



Sign in

Because you're accessing sensitive info, you need to verify your password.

willzhan@outlook.com

••••••••••

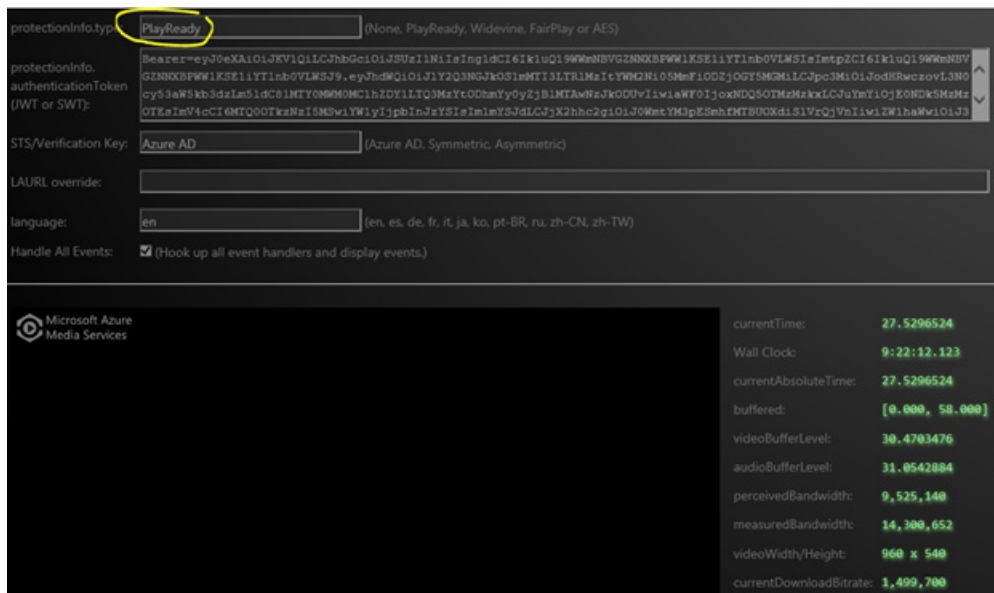
Sign in

Can't access your account?

Sign in with a different Microsoft account

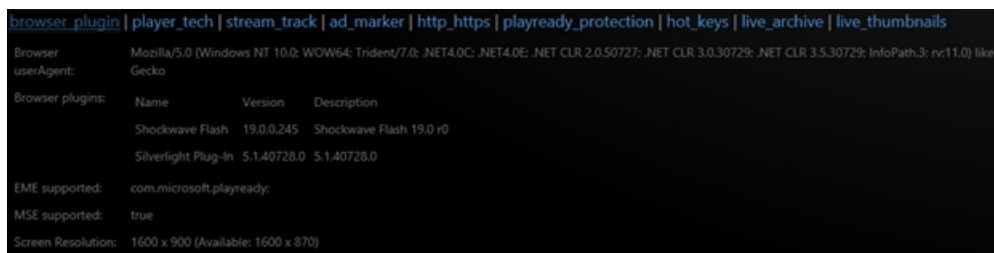
Use Encrypted Media Extensions for PlayReady

On a modern browser with Encrypted Media Extensions (EME) for PlayReady support, such as Internet Explorer 11 on Windows 8.1 or later and Microsoft Edge browser on Windows 10, PlayReady is the underlying DRM for EME.



The dark player area is because PlayReady protection prevents you from making a screen capture of protected video.

The following screenshot shows the player plug-ins and Microsoft Security Essentials (MSE)/EME support:



EME in Microsoft Edge and Internet Explorer 11 on Windows 10 allows [PlayReady SL3000](#) to be invoked on Windows 10 devices that support it. PlayReady SL3000 unlocks the flow of enhanced premium content (4K, HDR) and new content delivery models (for enhanced content).

To focus on the Windows devices, PlayReady is the only DRM in the hardware available on Windows devices (PlayReady SL3000). A streaming service can use PlayReady through EME or through a Universal Windows

Platform application and offer a higher video quality by using PlayReady SL3000 than another DRM. Typically, content up to 2K flows through Chrome or Firefox, and content up to 4K flows through Microsoft Edge/Internet Explorer 11 or a Universal Windows Platform application on the same device. The amount depends on service settings and implementation.

Use EME for Widevine

On a modern browser with EME/Widevine support, such as Chrome 41+ on Windows 10, Windows 8.1, Mac OSX Yosemite, and Chrome on Android 4.4.4, Google Widevine is the DRM behind EME.

The screenshot shows a video player interface with a dark theme. At the top, there's a dropdown menu for 'protectionInfo.type:' with 'Widevine' selected. Below it, the 'protectionInfo.authenticationToken (JWT or SWT):' field contains a long Base64-encoded string. The 'STS/Verification Key:' field is set to 'Azure AD'. The 'LAURL override:' field is empty. The 'language:' field is set to 'en'. The 'Handle All Events:' checkbox is checked. Below these settings, a video player shows a live performance of a band. To the right of the video, a statistics panel displays various metrics:

Metric	Value
currentTime:	27.176000
Wall Clock:	9:28:28.284
currentAbsoluteTime:	27.176
buffered:	[0.000, 66.664]
videoBufferLevel:	40.824000
audioBufferLevel:	39.488480
perceivedBandwidth:	14,974,316
measuredBandwidth:	17,909,003
videoWidth/Height:	1920 x 1080
currentDownloadBitrate:	6,011,366
currentPlaybackBitrate:	6,011,366

Widevine doesn't prevent you from making a screen capture of protected video.

The screenshot shows a browser's developer console with the 'browser_plugin' tab selected. It displays a list of browser plugins and their descriptions:

Name	Version	Description
Widevine Content Decryption Module		Enables Widevine licenses for playback of HTML audio/video content. (version: 1.4.8.866)
Chrome PDF Viewer		
Shockwave Flash	Shockwave Flash 19.0 r0	
Native Client		
Chrome PDF Viewer		Portable Document Format

Below the plugin list, it shows 'EME supported: true' and 'MSE supported: true'. At the bottom, it displays 'Screen Resolution: 1600 x 900 (Available: 1600 x 870)'.

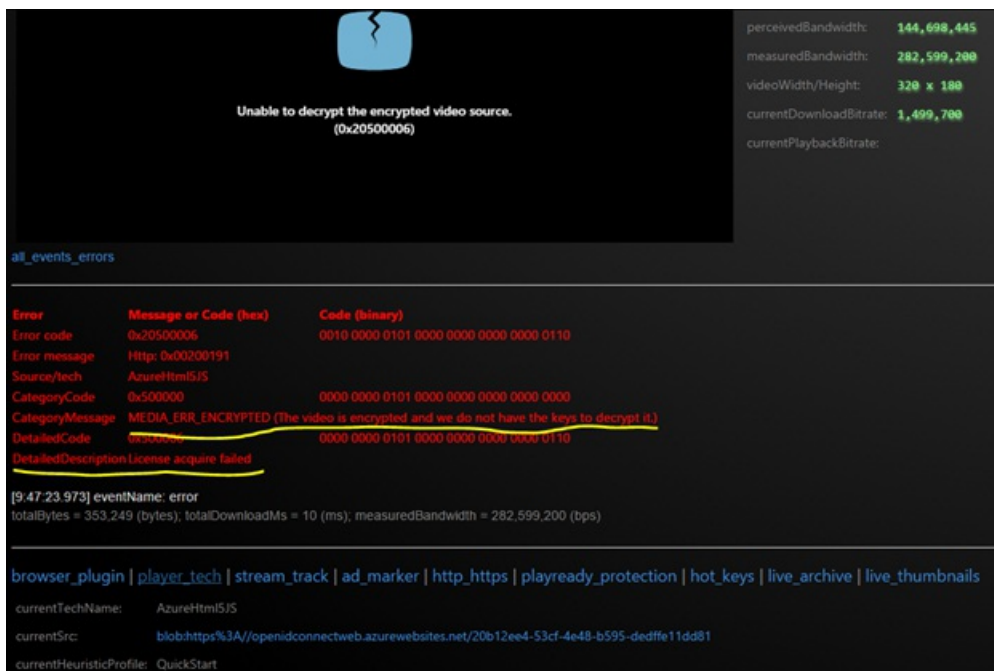
Use EME for FairPlay

Similarly, you can test FairPlay protected content in this test player in Safari on macOS or iOS 11.2 and later.

Make sure you put "FairPlay" as protectionInfo.type and put in the right URL for your Application Certificate in FPS AC Path (FairPlay Streaming Application Certificate Path).

Unentitled users

If a user isn't a member of the "Entitled Users" group, the user doesn't pass the entitlement check. The multi-DRM license service then refuses to issue the requested license as shown. The detailed description is "License acquire failed," which is as designed.



In both of the previous cases, user authentication stays the same. It takes place through Azure AD. The only difference is that JWTs are issued by the custom STS instead of Azure AD. When you configure dynamic CENC protection, the license delivery service restriction specifies the type of JWT, either a symmetric or an asymmetric key.

Summary

This document discussed content protection using 3 DRMs and access control via token authentication, its design, and its implementation by using Azure, Azure Media Services, and Azure Media Player.

- A reference design was presented that contains all the necessary components in a DRM subsystem.
- A reference implementation was presented on Azure, Azure Media Services, and Azure Media Player.
- Some topics directly involved in the design and implementation were also discussed.

Next steps

[Protect your content with DRM](#)

Live streaming with Azure Media Services v3

12/6/2018 • 7 minutes to read • [Edit Online](#)

Azure Media Services enables you to deliver live events to your customers on the Azure cloud. To stream your live events with Media Services, you need the following:

- A camera that is used to capture the live event.
- A live video encoder that converts signals from the camera (or another device, like a laptop) into a contribution feed that is sent to Media Services. The contribution feed can include signals related to advertising, such as SCTE-35 markers.
- Components in Media Services, which enable you to ingest, preview, package, record, encrypt, and broadcast the live event to your customers, or to a CDN for further distribution.

This article gives a detailed overview, guidance, and includes diagrams of the main components involved in live streaming with Media Services.

Overview of main components

To deliver on-demand or live streams with Media Services, you need to have at least one [StreamingEndpoint](#). When your Media Services account is created a **default** StreamingEndpoint is added to your account in the **Stopped** state. You need to start the StreamingEndpoint from which you want to stream your content to your viewers. You can use the default **StreamingEndpoint**, or create another customized **StreamingEndpoint** with your desired configuration and CDN settings. You may decide to enable multiple StreamingEndpoints, each one targeting a different CDN, and providing a unique hostname for delivery of content.

In Media Services, [LiveEvents](#) are responsible for ingesting and processing the live video feeds. When you create a LiveEvent, an input endpoint is created that you can use to send a live signal from a remote encoder. The remote live encoder sends the contribution feed to that input endpoint using either the [RTMP](#) or [Smooth Streaming](#) (fragmented-MP4) protocol.

Once the **LiveEvent** starts receiving the contribution feed, you can use its preview endpoint (preview URL to preview and validate that you are receiving the live stream before further publishing. After you have checked that the preview stream is good, you can use the LiveEvent to make the live stream available for delivery through one or more (pre-created) **StreamingEndpoints**. To accomplish this, you create a new [LiveOutput](#) on the **LiveEvent**.

The **LiveOutput** object is like a tape recorder that will catch and record the live stream into an Asset in your Media Services account. The recorded content will be persisted into the Azure Storage account attached to your account, into the container defined by the Asset resource. The **LiveOutput** also allows you to control some properties of the outgoing live stream, such as how much of the stream is kept in the archive recording (for example, the capacity of the cloud DVR). The archive on disk is a circular archive "window" that only holds the amount of content that is specified in the **archiveWindowLength** property of the **LiveOutput**. Content that falls outside of this window is automatically discarded from the storage container, and is not recoverable. You can create multiple LiveOutputs (up to three maximum) on a LiveEvent with different archive lengths and settings.

With Media Services you can take advantage of **Dynamic Packaging**, which allows you to preview and broadcast your live streams in [MPEG DASH](#), [HLS](#), and [Smooth Streaming formats](#) from the contribution feed that you send to the service. Your viewers can play back the live stream with any HLS, DASH, or Smooth Streaming compatible players. You can use [Azure Media Player](#) in your web or mobile applications to deliver your stream in any of these protocols.

Media Services enables you to deliver your content encrypted dynamically (**Dynamic Encryption**) with

Advanced Encryption Standard (AES-128) or any of the three major digital rights management (DRM) systems: Microsoft PlayReady, Google Widevine, and Apple FairPlay. Media Services also provides a service for delivering AES keys and DRM licenses to authorized clients. For more information on how to encrypt your content with Media Services, see [Protecting content overview](#)

If desired, you can also apply Dynamic Filtering, which can be used to control the number of tracks, formats, bitrates, and presentation time windows that are sent out to the players.

New capabilities for live streaming in v3

With the v3 APIs of Media Services, you benefit from the following new features:

- New low latency mode. For more information, see [latency](#).
- Improved RTMP support (increased stability and more source encoder support).
- RTMPS secure ingest.

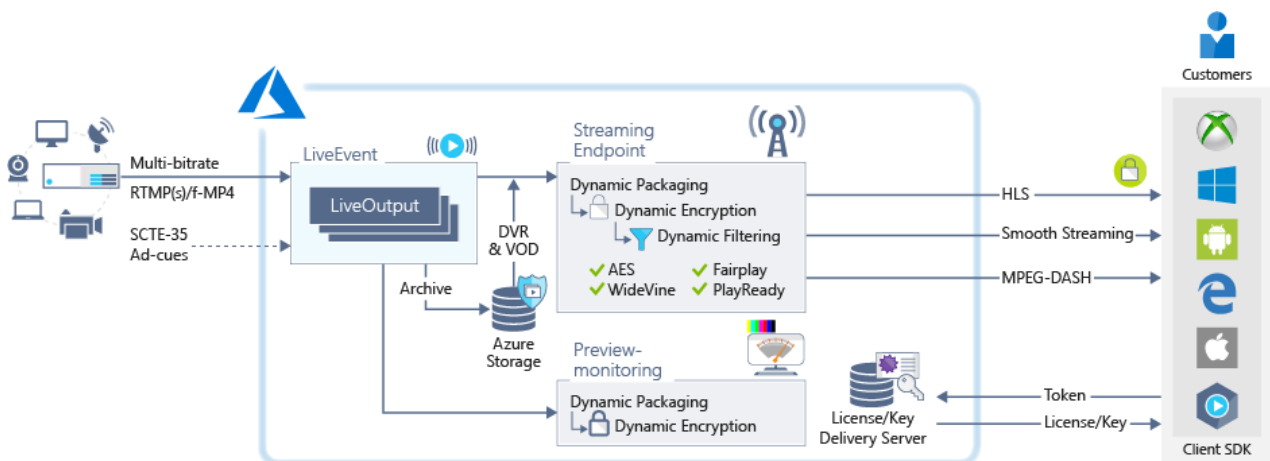
When you create a LiveEvent, you get 4 ingest URLs. The 4 ingest URLs are almost identical, have the same streaming token (AppId), only the port number part is different. Two of the URLs are primary and backup for RTMPS.

- You can stream live events that are up to 24 hours long when using Media Services for transcoding a single bitrate contribution feed into an output stream that has multiple bitrates.

LiveEvent types

A [LiveEvent](#) can be one of two types: pass-through and live encoding.

Pass-through



When using the pass-through LiveEvent, you rely on your on-premises live encoder to generate a multiple bitrate video stream and send that as the contribution feed to the LiveEvent (using RTMP or fragmented-MP4 protocol). The LiveEvent then carries through the incoming video streams without any further processing. Such a pass-through LiveEvent is optimized for long-running live events or 24x365 linear live streaming. When creating this type of LiveEvent, specify `None` (`LiveEventEncodingType.None`).

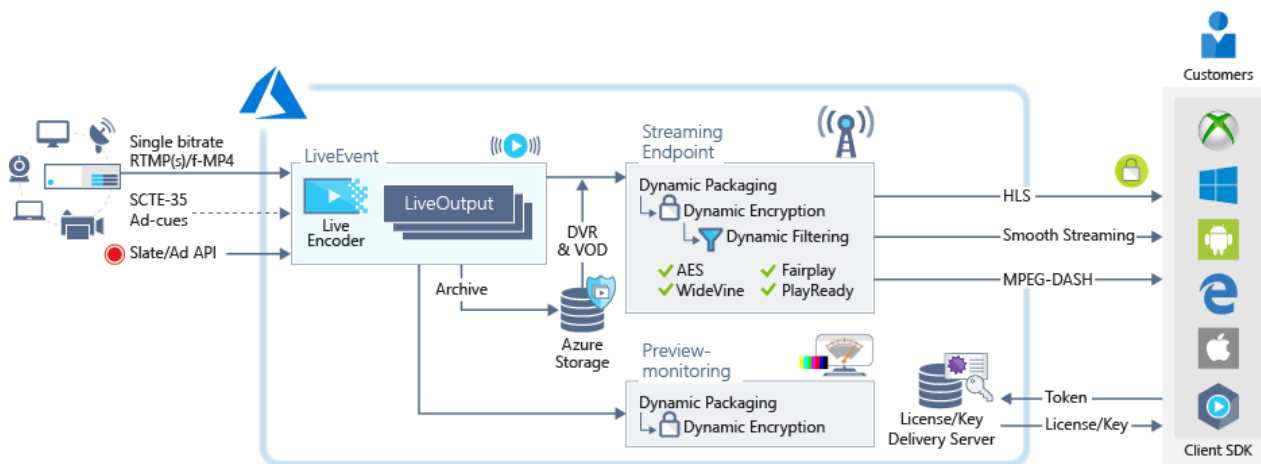
You can send the contribution feed at resolutions up to 4K and at a frame rate of 60 frames/second, with either H.264/AVC or H.265/HEVC video codecs, and AAC (AAC-LC, HE-AACv1, or HE-AACv2) audio codec. See the [LiveEvent types comparison and limitations](#) article for more details.

NOTE

Using a pass-through method is the most economical way to do live streaming when you are doing multiple events over a long period of time, and you have already invested in on-premises encoders. See [pricing](#) details.

See a live example in [MediaV3LiveApp](#).

Live encoding



When using live encoding with Media Services, you would configure your on-premises live encoder to send a single bitrate video as the contribution feed to the LiveEvent (using RTMP or Fragmented-Mp4 protocol). The LiveEvent encodes that incoming single bitrate stream to a [multiple bitrate video stream](#), makes it available for delivery to play back devices via protocols like MPEG-DASH, HLS, and Smooth Streaming. When creating this type of LiveEvent, specify the encoding type as **Basic** (LiveEventEncodingType.Basic).

You can send the contribution feed at up to 1080p resolution at a frame rate of 30 frames/second, with H.264/AVC video codec and AAC (AAC-LC, HE-AACv1, or HE-AACv2) audio codec. See the [LiveEvent types comparison and limitations](#) article for more details.

LiveEvent types comparison

The following article contains a table that compares features of the two LiveEvent types: [Comparison](#).

LiveOutput

A [LiveOutput](#) enables you to control the properties of the outgoing live stream, such as how much of the stream is recorded (for example, the capacity of the cloud DVR), and whether or not viewers can start watching the live stream. The relationship between a **LiveEvent** and its **LiveOutputs** relationship is similar to traditional television broadcast, whereby a channel (**LiveEvent**) represents a constant stream of video and a recording (**LiveOutput**) is scoped to a specific time segment (for example, evening news from 6:30PM to 7:00PM). You can record television using a Digital Video Recorder (DVR) – the equivalent feature in LiveEvents is managed via the ArchiveWindowLength property. It is an ISO-8601 timespan duration (for example, PTHH:MM:SS), which specifies the capacity of the DVR, and can be set from a minimum of 3 minutes to a maximum of 25 hours.

NOTE

LiveOutputs start on creation and stop when deleted. When you delete the **LiveOutput**, you are not deleting the underlying **Asset** and content in the Asset.

For more information, see [Using cloud DVR](#).

StreamingEndpoint

Once you have the stream flowing into the **LiveEvent**, you can begin the streaming event by creating an **Asset**, **LiveOutput**, and **StreamingLocator**. This will archive the stream and make it available to viewers through the [StreamingEndpoint](#).

When your Media Services account is created a default streaming endpoint is added to your account in the Stopped state. To start streaming your content and take advantage of dynamic packaging and dynamic encryption,

the streaming endpoint from which you want to stream content has to be in the Running state.

LiveEvent states and billing

A LiveEvent begins billing as soon as its state transitions to **Running**. To stop the LiveEvent from billing, you have to stop the LiveEvent.

For detailed information, see [States and billing](#).

Latency

For detailed information about LiveEvents latency, see [Latency](#).

Live streaming workflow

Here are the steps for a live streaming workflow:

1. Create a LiveEvent.
2. Create a new Asset object.
3. Create a LiveOutput and use the asset name that you created.
4. Create a Streaming Policy and Content Key if you intend to encrypt your content with DRM.
5. If not using DRM, create a Streaming Locator with the built-in Streaming Policy types.
6. List the paths on the Streaming Policy to get back the URLs to use (these are deterministic).
7. Get the hostname for the Streaming Endpoint you wish to stream from.
8. Combine the URL from step 6 with the hostname in step 7 to get your full URL.

For more information, see a [Live streaming tutorial](#) that is based on the [Live .NET Core](#) sample.

Next steps

- [LiveEvent types comparison](#)
- [States and billing](#)
- [Latency](#)

Recommended live streaming encoders

12/17/2018 • 3 minutes to read • [Edit Online](#)

In Media Services, a [LiveEvent](#) (channel) represents a pipeline for processing live-streaming content. A LiveEvent receives live input streams in one of two ways:

- An on-premises live encoder sends a multi-bitrate RTMP or Smooth Streaming (fragmented MP4) stream to the LiveEvent that is not enabled to perform live encoding with Media Services. The ingested streams pass through LiveEvents without any further processing. This method is called **pass-through**. A live encoder can send a single-bitrate stream to a pass through channel, but this configuration is not recommended because it does not allow for adaptive bitrate streaming to the client.

NOTE

Using a pass-through method is the most economical way to do live streaming.

- An on-premises live encoder sends a single-bitrate stream to the LiveEvent that is enabled to perform live encoding with Media Services in one of the following formats: RTMP or Smooth Streaming (fragmented MP4). The LiveEvent then performs live encoding of the incoming single-bitrate stream to a multi-bitrate (adaptive) video stream.

For detailed information about live encoding with Media Services, see [Live streaming with Media Services v3](#).

Live encoders that output RTMP

Media Services recommends using one of following live encoders that have RTMP as output:

- Adobe Flash Media Live Encoder 3.2
- Haivision KB
- Haivision Makito X HEVC
- OBS Studio
- Switcher Studio (iOS)
- Telestream Wirecast 8.1 +
- Telestream Wirecast S
- Teradek Slice 756
- TriCaster 8000
- Tricaster Mini HD-4
- VMIX
- xStream

Live encoders that output fragmented-MP4

Media Services recommends using one of the following live encoders that have multi-bitrate fragmented-MP4 (Smooth Streaming) as output:

- Ateme TITAN Live
- Cisco Digital Media Encoder 2200
- Elemental Live
- Envivio 4Caster C4 Gen III

- Imagine Communications Selenio MCP3
- Media Excel Hero Live and Hero 4K (UHD/HEVC)

How to become an on-premises encoder partner

As an Azure Media Services on-premises encoder partner, Media Services promotes your product by recommending your encoder to enterprise customers. To become an on-premises encoder partner, you must verify compatibility of your on-premises encoder with Media Services. To do so, complete the following verifications:

Pass-through LiveEvent verification

1. Create or visit your Azure Media Services account.
2. Create and start a **pass-through** LiveEvent.
3. Configure your encoder to push a multi-bitrate live stream.
4. Create a published live event.
5. Run your live encoder for approximately 10 minutes.
6. Stop the live event.
7. Create, start a Streaming endpoint, use a player such as [Azure Media Player](#) to watch the archived asset to ensure that playback has no visible glitches for all quality levels (Or alternatively watch and validate via the Preview URL during the live session before step 6).
8. Record the Asset ID, published streaming URL for the live archive, and the settings and version used from your live encoder.
9. Reset the LiveEvent state after creating each sample.
10. Repeat steps 3 through 9 for all configurations supported by your encoder (with and without ad signaling/captions/different encoding speeds).

Live encoding LiveEvent verification

1. Create or visit your Azure Media Services account.
2. Create and start a **live encoding** LiveEvent.
3. Configure your encoder to push a single-bitrate live stream.
4. Create a published live event.
5. Run your live encoder for approximately 10 minutes.
6. Stop the live event.
7. Create, start a Streaming endpoint, use a player such as [Azure Media Player](#) to watch the archived asset to ensure that playback has no visible glitches for all quality levels (Or alternatively watch and validate via the Preview URL during the live session before step 6).
8. Record the Asset ID, published streaming URL for the live archive, and the settings and version used from your live encoder.
9. Reset the LiveEvent state after creating each sample.
10. Repeat steps 3 through 9 for all configurations supported by your encoder (with and without ad signaling/captions/various encoding speeds).

Longevity verification

1. Create or visit your Azure Media Services account.
2. Create and start a **pass-through** channel.
3. Configure your encoder to push a multi-bitrate live stream.
4. Create a published live event.
5. Run your live encoder for one week or longer.
6. Use a player such as [Azure Media Player](#) to watch the live streaming from time to time (or archived asset) to ensure that playback has no visible glitches.
7. Stop the live event.

8. Record the Asset ID, published streaming URL for the live archive, and the settings and version used from your live encoder.

Lastly, email your recorded settings and live archive parameters to Azure Media Services at amsstreaming@microsoft.com as a notification that all self-verification checks have passed. Also, include your contact information for any follow ups. You can contact the Azure Media Services team with any questions regarding this process.

Next steps

[Live streaming with Media Services v3](#)

Using a cloud DVR

12/14/2018 • 2 minutes to read • [Edit Online](#)

A [LiveOutput](#) enables you to control the properties of the outgoing live stream, such as how much of the stream is recorded (for example, the capacity of the cloud DVR), and whether or not viewers can start watching the live stream. The relationship between a **LiveEvent** and its **LiveOutputs** is similar to traditional television broadcast, whereby a channel (**LiveEvent**) represents a constant stream of video and a recording (**LiveOutput**) is scoped to a specific time segment (for example, evening news from 6:30PM to 7:00PM). You can record television using a Digital Video Recorder (DVR) – the equivalent feature in LiveEvents is managed via the `ArchiveWindowLength` property. It is an ISO-8601 timespan duration (for example, PTHH:MM:SS), which specifies the capacity of the DVR, and can be set from a minimum of 3 minutes to a maximum of 25 hours.

LiveOutput

The **ArchiveWindowLength** value determines how far back in time a viewer can seek from the current live position. **ArchiveWindowLength** also determines how long the client manifests can grow.

Suppose you are streaming a football game, and it has an **ArchiveWindowLength** of only 30 minutes. A viewer who starts watching your event 45 minutes after the game started can seek back to at most the 15-minute mark. Your **LiveOutputs** for the game will continue until the **LiveEvent** is stopped, but content that falls outside of **ArchiveWindowLength** is continuously discarded from storage and is non-recoverable. In this example, the video between the start of the event and the 15-minute mark would have been purged from your DVR and from the container in blob storage for the [Asset](#). The archive is not recoverable and is removed from the container in Azure blob storage.

Each **LiveOutput** is associated with an **Asset**, which it uses to record the video into the associated Azure blob storage container. To publish the LiveOutput, you must create a **StreamingLocator** for that **Asset**. After creating a [StreamingLocator](#), you can build a streaming URL that you can provide to your viewers.

A **LiveEvent** supports up to three concurrently running **LiveOutputs** so you can create at most 3 recordings/archives from one live stream. This allows you to publish and archive different parts of an event as needed. Suppose you need to broadcast a 24x7 live linear feed, and create "recordings" of the different programs throughout the day to offer to customers as on-demand content for catch-up viewing. For this scenario, you first create a primary LiveOutput, with a short archive window of 1 hour or less – this is the primary live stream that your viewers would tune into. You would create a **StreamingLocator** for this **LiveOutput** and publish it to your application or web site as the "Live" feed. While the **LiveEvent** is running, you can programmatically create a second concurrent **LiveOutput** at the beginning of a program (or 5 minutes early to provide some handles to trim later). This second **LiveOutput** can be deleted 5 minutes after the program ends. With this second **Asset**, you can create a new **StreamingLocator** to publish this program as an on-demand asset in your application's catalog. You can repeat this process multiple times for other program boundaries or highlights that you wish to share as on-demand videos, all while the "Live" feed from the first **LiveOutput** continues to broadcast the linear feed.

NOTE

LiveOutputs start on creation and stop when deleted. When you delete the **LiveOutput**, you are not deleting the underlying **Asset** and content in the Asset.

Next steps

- [Live streaming overview](#)

- [Live streaming tutorial](#)

LiveEvent types comparison

12/6/2018 • 2 minutes to read • [Edit Online](#)

In Azure Media Services, a [LiveEvent](#) can be one of two types: live encoding and pass-through.

Types comparison

The following table compares features of the two LiveEvent types.

FEATURE	PASS-THROUGH LIVEEVENT	STANDARD (BASIC) LIVEEVENT
Single bitrate input is encoded into multiple bitrates in the cloud	No	Yes
Maximum video resolution for contribution feed	4K (4096x2160 at 60 frames/sec)	1080p (1920x1088 at 30 frames/sec)
Recommended maximum layers in contribution feed	Up to 12	One audio
Maximum layers in output	Same as input	Up to 7
Maximum aggregate bandwidth of contribution feed	60 Mbps	N/A
Maximum bitrate for a single layer in the contribution	20 Mbps	20 Mbps
Support for multiple language audio tracks	Yes	No
Supported input video codecs	H.264/AVC and H.265/HEVC	H.264/AVC
Supported output video codecs	Same as input	H.264/AVC
Supported video bit depth, input, and output	Up to 10-bit including HDR 10/HLG	8-bit
Supported input audio codecs	AAC-LC, HE-AAC v1, HE-AAC v2	AAC-LC, HE-AAC v1, HE-AAC v2
Supported output audio codecs	Same as input	AAC-LC
Maximum video resolution of output video	Same as input	720p (at 30 frames/second)
Input protocols	RTMP, fragmented-MP4 (Smooth Streaming)	RTMP, fragmented-MP4 (Smooth Streaming)
Price	See the pricing page and click on "Live Video" tab	See the pricing page and click on "Live Video" tab

FEATURE	PASS-THROUGH LIVEEVENT	STANDARD (BASIC) LIVEEVENT
Maximum run time	24x365 live linear	24x7
Ability to pass through embedded CEA 608/708 captions data	Yes	Yes
Support for inserting slates	No	No
Support for ad signaling via API	No	No
Support for ad signaling via SCTE-35 in-band messages	Yes	Yes
Ability to recover from brief stalls in contribution feed	Yes	No (LiveEvent will begin slating after 6+ seconds w/o input data)
Support for non-uniform input GOPs	Yes	No – input must have fixed GOP duration
Support for variable frame rate input	Yes	No – input must be fixed frame rate. Minor variations are tolerated, for example, during high motion scenes. But the contribution feed cannot drop the frame rate (for example, to 15 frames/sec).
Auto-shutoff of LiveEvent when input feed is lost	No	After 12 hours, if there is no LiveOutput running

Next steps

[Live streaming overview](#)

LiveEvent states and billing

11/28/2018 • 2 minutes to read • [Edit Online](#)

In Azure Media Services, a LiveEvent begins billing as soon as its state transitions to **Running**. To stop the LiveEvent from billing, you have to stop the LiveEvent.

When **LiveEventEncodingType** on your [LiveEvent](#) is set to Standard (Basic), Media Services auto shuts off any LiveEvent that is still in the **Running** state 12 hours after the input feed is lost, and there are no **LiveOutputs** running. However, you will still be billed for the time the LiveEvent was in the **Running** state.

States

The LiveEvent can be in one of the following states.

STATE	DESCRIPTION
Stopped	This is the initial state of the LiveEvent after creation (unless autostart was set to true.) No billing occurs in this state. In this state, the LiveEvent properties can be updated but streaming is not allowed.
Starting	The LiveEvent is being started and resources are being allocated. No billing occurs in this state. Updates or streaming are not allowed during this state. If an error occurs, the LiveEvent returns to the Stopped state.
Running	The LiveEvent resources have been allocated, ingest and preview URLs have been generated, and it is capable of receiving live streams. At this point, billing is active. You must explicitly call Stop on the LiveEvent resource to halt further billing.
Stopping	The LiveEvent is being stopped and resources are being de-provisioned. No billing occurs in this transient state. Updates or streaming are not allowed during this state.
Deleting	The LiveEvent is being deleted. No billing occurs in this transient state. Updates or streaming are not allowed during this state.

Next steps

- [Live streaming overview](#)
- [Live streaming tutorial](#)

LiveEvent latency in Media Services

11/30/2018 • 2 minutes to read • [Edit Online](#)

This article shows how to set low latency on a **LiveEvent**. It also discusses typical results that you see when using the low latency settings in various players. The results vary based on CDN and network latency.

To use the new **LowLatency** feature, you set the **StreamOptionsFlag** to **LowLatency** on the **LiveEvent**. Once the stream is up and running, you can use the [Azure Media Player](#) (AMP) demo page, and set the playback options to use the "Low Latency Heuristics Profile".

The following .NET example shows how to set **LowLatency** on the **LiveEvent**:

```
LiveEvent liveEvent = new LiveEvent(
    location: mediaService.Location,
    description: "Sample LiveEvent for testing",
    vanityUrl: false,
    encoding: new LiveEventEncoding(
        // Set this to Basic to enable a transcoding LiveEvent, and None to enable a pass-
through LiveEvent
        encodingType: LiveEventEncodingType.None,
        presetName: null
    ),
    input: new LiveEventInput(LiveEventInputProtocol.RTMP, liveEventInputAccess),
    preview: liveEventPreview,
    streamOptions: new List<StreamOptionsFlag?>()
    {
        // Set this to Default or Low Latency
        // To use low latency optimally, you should tune your encoder settings down to 1 second "Group
Of Pictures" (GOP) length instead of 2 seconds.
        StreamOptionsFlag.LowLatency
    }
);
```

See the full example: [MediaV3LiveApp](#).

LiveEvents latency

The following tables show typical results for latency (when the LowLatency flag is enabled) in Media Services, measured from the time the contribution feed reaches the service to when a viewer sees the playback on the player. To use low latency optimally, you should tune your encoder settings down to 1 second "Group Of Pictures" (GOP) length. When using a higher GOP length, you minimize bandwidth consumption and reduce bitrate under same frame rate. It is especially beneficial in videos with less motion.

Pass-through

	2S GOP LOW LATENCY ENABLED	1S GOP LOW LATENCY ENABLED
DASH in AMP	10s	8s
HLS on native iOS player	14s	10s

Live encoding

	2S GOP LOW LATENCY ENABLED	1S GOP LOW LATENCY ENABLED
DASH in AMP	14s	10s
HLS on native iOS player	18s	13s

NOTE

The end-to-end latency can vary depending on local network conditions or by introducing a CDN caching layer. You should test your exact configurations.

Next steps

- [Live streaming overview](#)
- [Live streaming tutorial](#)

Create an Azure Media Services account

12/10/2018 • 2 minutes to read • [Edit Online](#)

To start encrypting, encoding, analyzing, managing, and streaming media content in Azure, you need to create a Media Services account. At the time, you create a Media Services account, you also create an associated storage account (or use an existing one) in the same geographic region as the Media Services account. Azure This article describes steps for creating a new Azure Media Services account using the Azure CLI.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- An active Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

Set the Azure subscription

In the following command, provide the Azure subscription ID that you want to use for the Media Services account. You can see a list of subscriptions that you have access to by navigating to [Subscriptions](#).

```
az account set --subscription mySubscriptionId
```

Create a Media Services account

You first need to create a Media Services account. This section shows what you need for the account creation using the Azure CLI.

Create a resource group

Create a resource group using the following command. An Azure resource group is a logical container into which resources like Azure Media Services accounts and the associated Storage accounts are deployed and managed.

```
az group create --name amsResourceGroup --location westus2
```

Create a storage account

When creating a Media Services account, you need to supply the name of an Azure Storage account resource. The specified storage account is attached to your Media Services account.

You must have one **Primary** storage account and you can have any number of **Secondary** storage accounts associated with your Media Services account. Media Services supports **General-purpose v2** (GPv2) or **General-purpose v1** (GPv1) accounts. Blob only accounts are not allowed as **Primary**. If you want to learn more about storage accounts, see [Azure Storage account options](#).

For more information about how storage accounts are used in Media Services, see [Storage accounts](#).

The following command creates a Storage account that is going to be associated with the Media Services account. In the script below, you can substitute `storageaccountforams` with your value. The account name must

have length less than 24.

```
az storage account create --name storageaccountforams \  
--kind StorageV2 \  
--sku Standard_RAGRS \  
--resource-group amsResourceGroup
```

Create a Media Services account

The following Azure CLI command creates a new Media Services account. You can replace the following values:

`amsaccount` `storageaccountforams` (must match the value you gave for your storage account), and
`amsResourceGroup` (must match the value you gave for the resource group).

```
az ams account create --name amsaccount --resource-group amsResourceGroup --storage-account  
storageaccountforams
```

Next steps

[Stream a file](#)

See also

[Azure CLI](#)

Access Azure Media Services API with the Azure CLI

11/13/2018 • 2 minutes to read • [Edit Online](#)

You should use the Azure AD service principal authentication to connect to the Azure Media Services API. Your application needs to request an Azure AD token that has the following parameters:

- Azure AD tenant endpoint
- Media Services resource URI
- Resource URI for REST Media Services
- Azure AD application values: the client ID and client secret

This article shows you how to use the Azure CLI to create an Azure AD application and service principal and get the values that are needed to access Azure Media Services resources.

Prerequisites

- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#).

Make sure to remember the values that you used for the resource group name and Media Services account name.

Access the Media Services API

To connect to Azure Media Services APIs, you use the Azure AD service principal authentication. The following command creates an Azure AD application and attaches a service principal to the account. You should use the returned values to configure your application.

Before running the script, you can replace the `amsaccount` and `amsResourceGroup` with the names you chose when creating these resources. `amsaccount` is the name of the Azure Media Services account where to attach the service principal.

The following command returns a `json` output:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup
```

This command produces a response similar to this:

```
{
  "AadClientId": "00000000-0000-0000-0000-000000000000",
  "AadEndpoint": "https://login.microsoftonline.com",
  "AadSecret": "00000000-0000-0000-0000-000000000000",
  "AadTenantId": "00000000-0000-0000-0000-000000000000",
  "AccountName": "amsaccount",
  "ArmAadAudience": "https://management.core.windows.net/",
  "ArmEndpoint": "https://management.azure.com/",
  "Region": "West US 2",
  "ResourceGroup": "amsResourceGroup",
  "SubscriptionId": "00000000-0000-0000-0000-000000000000"
}
```

If you would like to get an `xml` in the response, use the following command:

```
az ams account sp create --account-name amsaccount --resource-group amsResourceGroup --xml
```

Next steps

[Stream a file](#)

See also

[Azure CLI](#)

Create a job input from an HTTPS URL

12/14/2018 • 2 minutes to read • [Edit Online](#)

In Media Services v3, when you submit Jobs to process your videos, you have to tell Media Services where to find the input video. One of the options is to specify an HTTP(s) URL as a job input (as shown in this example). Note that currently, AMS v3 does not support chunked transfer encoding over HTTPS URLs. For a full example, see this [GitHub sample](#).

.NET sample

The following code shows how to create a job with an HTTPS URL input.

```
private static async Task<Job> SubmitJobAsync(IAzureMediaServicesClient client,
    string resourceGroup,
    string accountName,
    string transformName,
    string outputAssetName,
    string jobName)
{
    // This example shows how to encode from any HTTPS source URL - a new feature of the v3 API.
    // Change the URL to any accessible HTTPS URL or SAS URL from Azure.
    JobInputHttp jobInput =
        new JobInputHttp(files: new[] { "https://nimbuscdn-
nimbuspm.streaming.mediaservices.windows.net/2b533311-b215-4409-80af-529c3e853622/ignite-short.mp4" });

    JobOutput[] jobOutputs =
    {
        new JobOutputAsset(outputAssetName),
    };

    // In this example, we are assuming that the job name is unique.
    //
    // If you already have a job with the desired name, use the Jobs.Get method
    // to get the existing job. In Media Services v3, Get methods on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).
    Job job = await client.Jobs.CreateAsync(
        resourceGroup,
        accountName,
        transformName,
        jobName,
        new Job
        {
            Input = jobInput,
            Outputs = jobOutputs,
        });

    return job;
}
```

Next steps

[Create a job input from a local file.](#)

Create a job input from a local file

12/14/2018 • 2 minutes to read • [Edit Online](#)

In Media Services v3, when you submit Jobs to process your videos, you have to tell Media Services where to find the input video. The input video can be stored as a Media Service Asset, in which case you create an input asset based on a file (stored locally or in Azure Blob storage). This topic shows how to create a job input from a local file. For a full example, see this [GitHub sample](#).

.NET sample

The following code shows how to create an input asset and use it as the input for the job. The `CreateInputAsset` function performs the following actions:

- Creates the Asset
- Gets a writable [SAS URL](#) to the Asset's [container in storage](#)
- Uploads the file into the container in storage using the SAS URL

```

private static async Task<Asset> CreateInputAssetAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string assetName,
    string fileToUpload)
{
    // In this example, we are assuming that the asset name is unique.
    //
    // If you already have an asset with the desired name, use the Assets.Get method
    // to get the existing asset. In Media Services v3, the Get method on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).

    // Call Media Services API to create an Asset.
    // This method creates a container in storage for the Asset.
    // The files (blobs) associated with the asset will be stored in this container.
    Asset asset = await client.Assets.CreateOrUpdateAsync(resourceGroupName, accountName, assetName, new
Asset());

    // Use Media Services API to get back a response that contains
    // SAS URL for the Asset container into which to upload blobs.
    // That is where you would specify read-write permissions
    // and the expiration time for the SAS URL.
    var response = await client.Assets.ListContainerSasAsync(
        resourceGroupName,
        accountName,
        assetName,
        permissions: AssetContainerPermission.ReadWrite,
        expiryTime: DateTime.UtcNow.AddHours(4).ToUniversalTime());

    var sasUri = new Uri(response.AssetContainerSasUrls.First());

    // Use Storage API to get a reference to the Asset container
    // that was created by calling Asset's CreateOrUpdate method.
    CloudBlobContainer container = new CloudBlobContainer(sasUri);
    var blob = container.GetBlockBlobReference(Path.GetFileName(fileToUpload));

    // Use Strorage API to upload the file into the container in storage.
    await blob.UploadFromFileAsync(fileToUpload);

    return asset;
}

```

```

private static async Task<Job> SubmitJobAsync(IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string jobName,
    string inputAssetName,
    string outputAssetName)
{
    // Use the name of the created input asset to create the job input.
    JobInput jobInput = new JobInputAsset(assetName: inputAssetName);

    JobOutput[] jobOutputs =
    {
        new JobOutputAsset(outputAssetName),
    };

    // In this example, we are assuming that the job name is unique.
    //
    // If you already have a job with the desired name, use the Jobs.Get method
    // to get the existing job. In Media Services v3, the Get method on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).
    Job job = await client.Jobs.CreateAsync(
        resourceGroupName,
        accountName,
        transformName,
        jobName,
        new Job
        {
            Input = jobInput,
            Outputs = jobOutputs,
        });

    return job;
}

```

Next steps

[Create a job input from an HTTPS URL.](#)

Upload files into a Media Services account using REST

12/19/2018 • 2 minutes to read • [Edit Online](#)

In Media Services, you upload your digital files into a blob container associated with an asset. The [Asset](#) entity can contain video, audio, images, thumbnail collections, text tracks and closed caption files (and the metadata about these files). Once the files are uploaded into the asset's container, your content is stored securely in the cloud for further processing and streaming.

This article shows you how to upload a local file using REST.

Prerequisites

To complete the steps described in this topic, you have to:

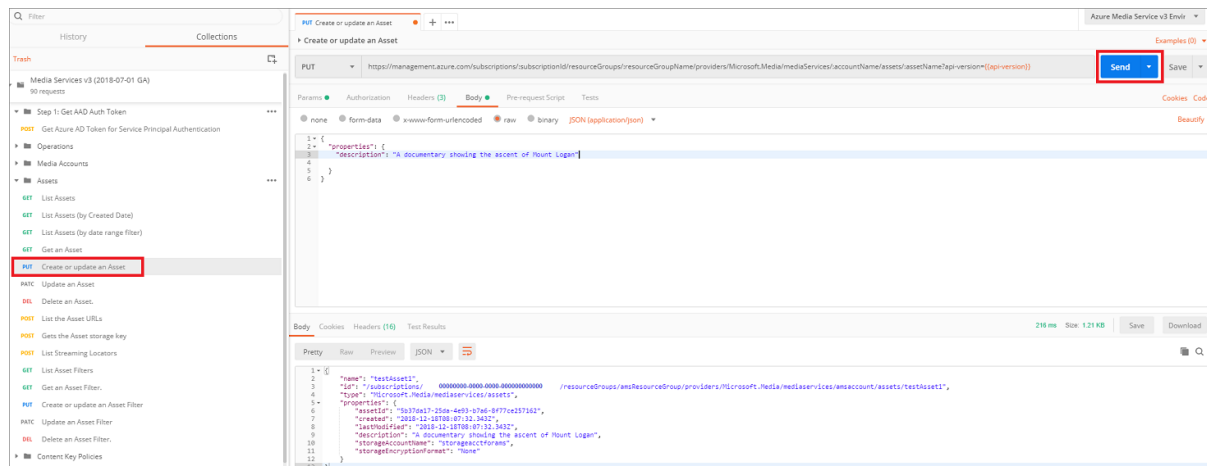
- Review [Asset concept](#).
- [Configure Postman for Azure Media Services REST API calls](#).

Make sure to follow the last step in the topic [Get Azure AD Token](#).

Create an asset

This section shows how to create a new Asset.

1. Select **Assets** -> **Create or update an Asset**.
2. Press **Send**.

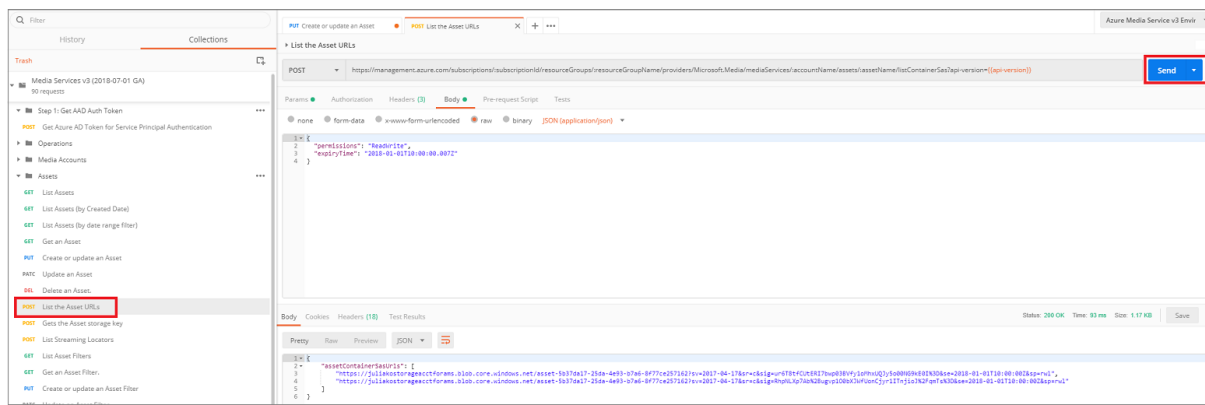


You see the **Response** with the info about newly created asset.

Get a SAS URL with read-write permissions

This section shows how to get a SAS URL that was generated for the created asset. The SAS URL was created with read-write permissions and can be used to upload digital files into the Asset container.

1. Select **Assets** -> **List the Asset URLs**.
2. Press **Send**.



You see the **Response** with the info about asset's URLs. Copy the first URL and use it to upload your file.

Upload a file to blob storage using the upload URL

Use the Azure Storage APIs or SDKs (for example, the [Storage REST API](#), [JAVA SDK](#), or [.NET SDK](#)).

Next steps

[Tutorial: Encode a remote file based on URL and stream the video - REST](#)

Create filters with Media Services .NET SDK

12/19/2018 • 2 minutes to read • [Edit Online](#)

When delivering your content to customers (streaming Live events or Video on Demand) your client might need more flexibility than what's described in the default asset's manifest file. Azure Media Services enables you to define account filters and asset filters for your content. For more information, see [Filters and dynamic manifests](#).

This topic shows how to use Media Services .NET SDK to define a filter for a Video on Demand asset and create [Account Filters](#) and [Asset Filters](#).

Prerequisites

- Review [Filters and dynamic manifests](#).
- [Create a Media Services account](#). Make sure to remember the resource group name and the Media Services account name.
- Get information needed to [access APIs](#)
- Review [Upload, encode, and stream using Azure Media Services](#) to see how to [start using .NET SDK](#)

Define a filter

In .NET, you configure track selections with [FilterTrackSelection](#) and [FilterTrackPropertyCondition](#) classes.

The following code defines a filter that includes any audio tracks that are EC-3 and any video tracks that have bitrate in the 0-1000000 range.

```
var audioConditions = new List<FilterTrackPropertyCondition>()
{
    new FilterTrackPropertyCondition(FilterTrackPropertyType.Type, "Audio",
    FilterTrackPropertyCompareOperation.Equal),
    new FilterTrackPropertyCondition(FilterTrackPropertyType.FourCC, "EC-3",
    FilterTrackPropertyCompareOperation.Equal)
};

var videoConditions = new List<FilterTrackPropertyCondition>()
{
    new FilterTrackPropertyCondition(FilterTrackPropertyType.Type, "Video",
    FilterTrackPropertyCompareOperation.Equal),
    new FilterTrackPropertyCondition(FilterTrackPropertyType.Bitrate, "0-1000000",
    FilterTrackPropertyCompareOperation.Equal)
};

List<FilterTrackSelection> includedTracks = new List<FilterTrackSelection>()
{
    new FilterTrackSelection(audioConditions),
    new FilterTrackSelection(videoConditions)
};
```

Create account filters

The following code shows how to use .NET to create an account filter that includes all track selections [defined above](#).

```
AccountFilter accountFilterParams = new AccountFilter(tracks: includedTracks);
client.AccountFilters.CreateOrUpdate(config.ResourceGroup, config.AccountName, "accountFilterName1",
accountFilter);
```

Create asset filters

The following code shows how to use .NET to create an asset filter that includes all track selections [defined above](#).

```
AssetFilter assetFilterParams = new AssetFilter(tracks: includedTracks);
client.AssetFilters.CreateOrUpdate(config.ResourceGroup, config.AccountName, encodedOutputAsset.Name,
"assetFilterName1", assetFilterParams);
```

Next steps

[Stream videos](#)

Creating filters with Media Services REST API

12/19/2018 • 2 minutes to read • [Edit Online](#)

When delivering your content to customers (streaming Live events or Video on Demand) your client might need more flexibility than what's described in the default asset's manifest file. Azure Media Services enables you to define account filters and asset filters for your content. For more information, see [Filters and dynamic manifests](#).

This topic shows how to define a filter for a Video on Demand asset and use REST APIs to create [Account Filters](#) and [Asset Filters](#).

Prerequisites

To complete the steps described in this topic, you have to:

- Review [Filters and dynamic manifests](#).
- [Configure Postman for Azure Media Services REST API calls](#).

Make sure to follow the last step in the topic [Get Azure AD Token](#).

Define a filter

The following is the **Request body** example that defines the track selection conditions that are added to the manifest. This filter includes any audio tracks that are EC-3 and any video tracks that have bitrate in the 0-1000000 range.

```
{
  "properties": {
    "tracks": [
      {
        "trackSelections": [
          {
            "property": "Type",
            "value": "Audio",
            "operation": "Equal"
          },
          {
            "property": "FourCC",
            "value": "EC-3",
            "operation": "Equal"
          }
        ]
      },
      {
        "trackSelections": [
          {
            "property": "Type",
            "value": "Video",
            "operation": "Equal"
          },
          {
            "property": "Bitrate",
            "value": "0-1000000",
            "operation": "Equal"
          }
        ]
      }
    ]
  }
}
```

Create account filters

In the Postman's collection that you downloaded, select **Account Filters**-> **Create or update an Account Filter**.

The **PUT** HTTP request method is similar to:

```
PUT
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Media/mediaServices/{accountName}/accountFilters/{filterName}?api-version=2018-07-01
```

Select the **Body** tab and paste the json code you [defined earlier](#).

Select **Send**.

The filter has been created.

For more information, see [Create or update](#). Also, see [JSON examples for filters](#).

Create asset filters

In the "Media Services v3" Postman collection that you downloaded, select **Assets**-> **Create or update Asset Filter**.

The **PUT** HTTP request method is similar to:

PUT

```
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Media/mediaServices/{accountName}/assets/{assetName}/assetFilters/{filterName}?api-version=2018-07-01
```

Select the **Body** tab and paste the json code you [defined earlier](#).

Select **Send**.

The asset filter has been created.

For details on how to create or update asset filters, see [Create or update](#). Also, see [JSON examples for filters](#).

Next steps

[Stream videos](#)

Creating filters with CLI

12/19/2018 • 2 minutes to read • [Edit Online](#)

When delivering your content to customers (streaming Live events or Video on Demand), your client might need more flexibility than what's described in the default asset's manifest file. Azure Media Services enables you to define account filters and asset filters for your content. For more information, see [Filters and dynamic manifests](#).

This topic shows how to configure a filter for a Video on-Demand asset and use CLI for Media Services v3 to create [Account Filters](#) and [Asset Filters](#).

Prerequisites

- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#). Make sure to remember the resource group name and the Media Services account name.
- Review [Filters and dynamic manifests](#).

Define a filter

The following example defines the track selection conditions that are added to the final manifest. This filter includes any audio tracks that are EC-3 and any video tracks that have bitrate in the 0-1000000 range.

Filters defined in REST, include the "Properties" wrapper JSON object.

```
[
  {
    "trackSelections": [
      {
        "property": "Type",
        "value": "Audio",
        "operation": "Equal"
      },
      {
        "property": "FourCC",
        "value": "EC-3",
        "operation": "NotEqual"
      }
    ]
  },
  {
    "trackSelections": [
      {
        "property": "Type",
        "value": "Video",
        "operation": "Equal"
      },
      {
        "property": "Bitrate",
        "value": "0-1000000",
        "operation": "Equal"
      }
    ]
  }
]
```

Create account filters

The following [az ams account-filter](#) command creates an account filter with filter track selections that were [defined earlier](#).

```
az ams account-filter create -a amsAccount -g resourceGroup -n filterName --tracks @C:\tracks.json
```

Also, see [JSON examples for filters](#).

Create asset filters

The following [az ams asset-filter](#) command creates an asset filter with filter track selections that were [defined earlier](#).

```
az ams asset-filter create -a amsAccount -g resourceGroup -n filterName --asset-name assetName --tracks @C:\tracks.json
```

Also, see [JSON examples for filters](#).

Next step

[Stream videos](#)

See also

[Azure CLI](#)

How to encode with a custom Transform

12/10/2018 • 3 minutes to read • [Edit Online](#)

When encoding with Azure Media Services, you can get started quickly with one of the recommended built-in presets based on industry best practices as demonstrated in the [Streaming files](#) tutorial, or you can choose to build a custom preset to target your specific scenario or device requirements.

NOTE

In Azure Media Services v3, all of the encoding bit rates are in bits per second. This is different than the REST v2 Media Encoder Standard presets. For example, the bitrate in v2 would be specified as 128, but in v3 it would be 128000.

Download the sample

Clone a GitHub repository that contains the full .NET Core sample to your machine using the following command:

```
git clone https://github.com/Azure-Samples/media-services-v3-dotnet-core-tutorials.git
```

The custom preset sample is located in the [EncodeCustomTransform](#) folder.

Create a transform with a custom preset

When creating a new [Transform](#), you need to specify what you want it to produce as an output. The required parameter is a **TransformOutput** object, as shown in the code below. Each **TransformOutput** contains a **Preset**. **Preset** describes the step-by-step instructions of video and/or audio processing operations that are to be used to generate the desired **TransformOutput**. The following **TransformOutput** creates custom codec and layer output settings.

When creating a [Transform](#), you should first check if one already exists using the **Get** method, as shown in the code that follows. In Media Services v3, **Get** methods on entities return **null** if the entity doesn't exist (a case-insensitive check on the name).

```
private static Transform EnsureTransformExists(IAzureMediaServicesClient client, string resourceGroupName,
string accountName, string transformName)
{
    // Does a Transform already exist with the desired name? Assume that an existing Transform with the
    // desired name
    // also uses the same recipe or Preset for processing content.
    Transform transform = client.Transforms.Get(resourceGroupName, accountName, transformName);

    if (transform == null)
    {
        // Create a new Transform Outputs array - this defines the set of outputs for the Transform
        TransformOutput[] outputs = new TransformOutput[]
        {
            // Create a new TransformOutput with a custom Standard Encoder Preset
            // This demonstrates how to create custom codec and layer output settings

            new TransformOutput(
                new StandardEncoderPreset(
                    codecs: new Codec[]
                    {
                        // Add a new H.264 video codec
                    }
                )
            )
        };

        // Create a new Transform with the specified name and outputs
        Transform transform2 = client.Transforms.Create(resourceGroupName, accountName, transformName, outputs);
    }
}
```

```

        // Add an AAC Audio layer for the audio encoding
        new AACAudio(
            channels: 2,
            samplingRate: 48000,
            bitrate: 128000,
            profile: AACAudioProfile.AACLC
        ),
        // Next, add a H264Video for the video encoding
        new H264Video (
            // Set the GOP interval to 2 seconds for both H264Layers
            keyFrameInterval: TimeSpan.FromSeconds(2),
            // Add H264Layers, one at HD and the other at SD. Assign a label that you can
            use for the output filename
            layers: new H264Layer[]
            {
                new H264Layer (
                    bitrate: 1000000, // Note that the units is in bits per second
                    width: "1280",
                    height: "720",
                    label: "HD" // This label is used to modify the file name in the output
                    formats
                ),
                new H264Layer (
                    bitrate: 600000,
                    width: "640",
                    height: "360",
                    label: "SD"
                )
            }
        ),
        // Also generate a set of PNG thumbnails
        new PngImage(
            start: "25%",
            step: "25%",
            range: "80%",
            layers: new PngLayer[]
            {
                new PngLayer(
                    width: "50%",
                    height: "50%"
                )
            }
        ),
        // Specify the format for the output files - one for video+audio, and another for the
        thumbnails
        formats: new Format[]
        {
            // Mux the H.264 video and AAC audio into MP4 files, using basename, label, bitrate
            and extension macros
            // Note that since you have multiple H264Layers defined above, you have to use a
            macro that produces unique names per H264Layer
            // Either {Label} or {Bitrate} should suffice

            new Mp4Format(
                filenamePattern: "Video-{Basename}-{Label}-{Bitrate}{Extension}"
            ),
            new PngFormat(
                filenamePattern: "Thumbnail-{Basename}-{Index}{Extension}"
            )
        }
    ),
    onError: OnErrorType.StopProcessingJob,
    relativePriority: Priority.Normal
);

string description = "A simple custom encoding transform with 2 MP4 bitrates";
// Create the custom Transform with the outputs defined above
transform = client.Transforms.CreateOrUpdate(resourceGroupName, accountName, transformName, outputs,

```

```
description);  
    }  
  
    return transform;  
}
```

Next steps

[Streaming files](#)

Create and monitor Media Services events with Event Grid using the Azure portal

10/17/2018 • 2 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use the Azure portal to subscribe to events for your Azure Media Services account. Then, you trigger events to view the result. Typically, you send events to an endpoint that processes the event data and takes actions. In this article, we send events to a web app that collects and displays the messages.

When you're finished, you see that the event data has been sent to the web app.

Prerequisites

- Have an active Azure subscription.
- Create a new Azure Media Services account, as described in [this quickstart](#).

Create a message endpoint

Before subscribing to the events for the Media Services account, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. In this article, you deploy a [pre-built web app](#) that displays the event messages. The deployed solution includes an App Service plan, an App Service web app, and source code from GitHub.

1. Select **Deploy to Azure** to deploy the solution to your subscription. In the Azure portal, provide values for the parameters.



2. The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: `https://<your-site-name>.azurewebsites.net`

If you switch to the "Azure Event Grid Viewer" site, you see it has no events yet.

Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider.

In the Azure portal:

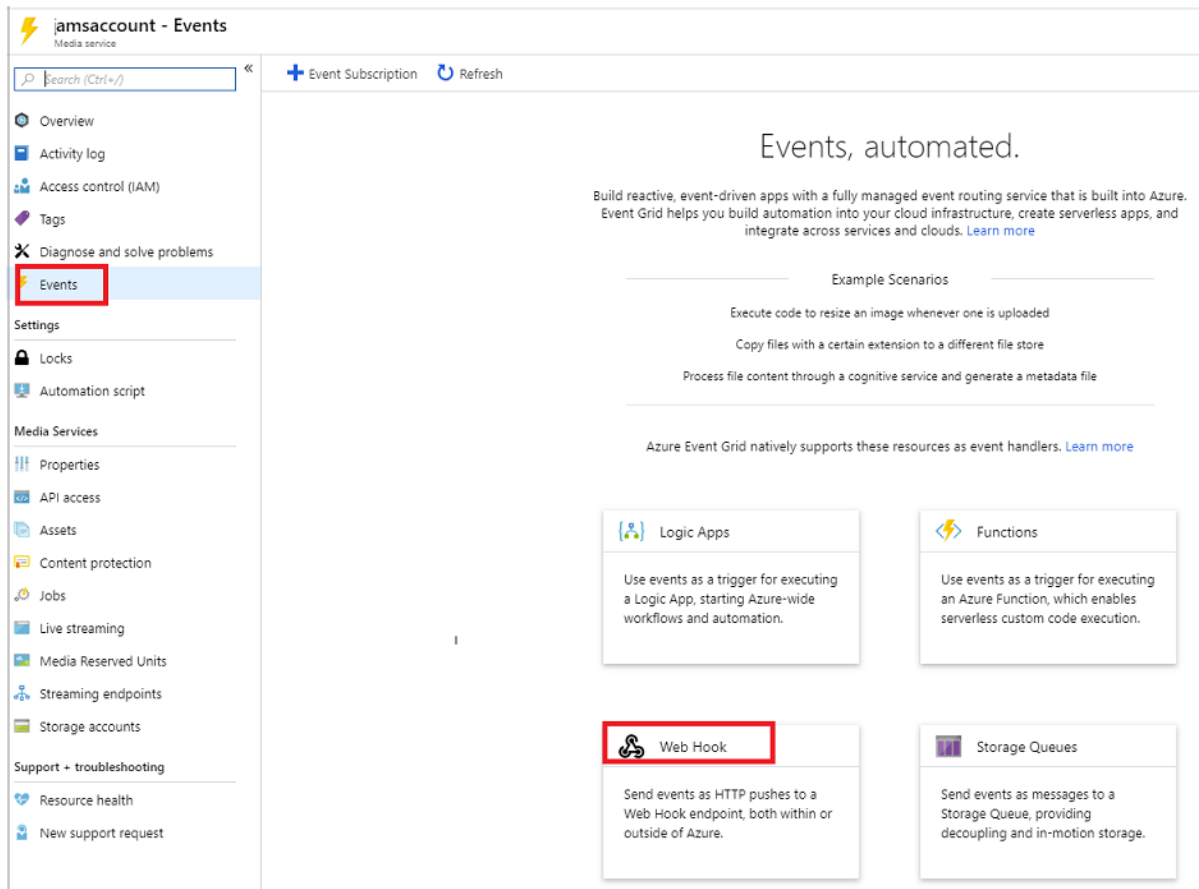
1. Select **Subscriptions**.
2. Select the subscription you're using for Event Grid.
3. Under **Settings**, select **Resource providers**.
4. Find **Microsoft.EventGrid**.
5. If not registered, select **Register**.

It may take a moment for the registration to finish. Select **Refresh** to update the status. When **Status** is **Registered**, you're ready to continue.

Subscribe to Media Services events

You subscribe to a topic to tell Event Grid which events you want to track, and where to send the events.

1. In the portal, select your Media Services account and select **Events**.
2. To send events to your viewer app, use a web hook for the endpoint.



3. The event subscription is prefilled with values for your Media Services account.
4. Select 'Web Hook' for the **Endpoint Type**.
5. In this topic, we leave the **Subscribe to all event types** checked. However, you can uncheck it and filter for specific event types.
6. Click on the **Select an endpoint** link.

For the web hook endpoint, provide the URL of your web app and add `api/updates` to the home page URL.

7. Press **Confirm Selection**.
8. Press **Create**.
9. Give your subscription a name.

The screenshot shows the Azure Event Grid 'Create Event Subscription' interface. The left panel, titled 'Create Event Subscription', has tabs for 'Basic' and 'Additional Features'. It includes sections for 'TOPIC DETAILS' (Topic Type: Media service, Topic Resource: amsaccount), 'EVENT TYPES' (Subscribe to all event types: checked), 'ENDPOINT DETAILS' (Endpoint Type: Web Hook, Endpoint: Select an endpoint), and 'EVENT SUBSCRIPTION DETAILS' (Name: amsdemosubscription, Event Schema: Event Grid Schema). A 'Create' button is at the bottom. The right panel, titled 'Select Web Hook', shows a 'Subscriber Endpoint' field with the URL 'https://jmsstestsite.azurewebsites.net/api/updates' and a 'Confirm Selection' button at the bottom.

10. View your web app again, and notice that a subscription validation event has been sent to it.

Event Grid sends the validation event so the endpoint can verify that it wants to receive event data. The endpoint has to set `validationResponse` to `validationCode`. For more information, see [Event Grid security and authentication](#). You can view the web app code to see how it validates the subscription.

Now, let's trigger events to see how Event Grid distributes the message to your endpoint.

Send an event to your endpoint













You can trigger events for the Media Services account by running an encoding job. You can follow [this quickstart](#) to encode a file and start sending events. If you subscribed to all events, you will see a screen similar to the following:

TIP

Select the eye icon to expand the event data. Do not refresh the page, if you want to view all the events.



Azure Event Grid Viewer

Event Type	Subject
 Microsoft.Media.JobFinished	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputFinished	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobProcessing	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputProcessing	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputScheduled	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobScheduled	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18

```
{
  "topic": "/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/amsResourceGroup/providers/Microsoft.Media/mediaservices/amservices/...",
  "subject": "transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18c",
  "eventType": "Microsoft.Media.JobOutputStateChange",
  "eventTime": "2018-10-13T08:36:56.264887",
  "id": "4c617372-baec-4f88-8352-65d83d2d3905",
  "data": {
    "previousState": "Queued",
    "output": {
      "@odata.type": "#Microsoft.Media.JobOutputAsset",
      "assetName": "output-8bad7d4c6ff843069f6db1e66cbce18c",
      "error": null,
      "label": "BuiltInStandardEncoderPreset_0",
      "progress": 0,
      "state": "Scheduled"
    },
    "jobCorrelationData": {}
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}
```

Next steps

[Upload, encode, and stream](#)

Create and monitor Media Services events with Event Grid using the Azure CLI

11/13/2018 • 3 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use the Azure CLI to subscribe to events for your Azure Media Services account. Then, you trigger events to view the result. Typically, you send events to an endpoint that processes the event data and takes actions. In this article, you send the events to a web app that collects and displays the messages.

Prerequisites

- An active Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#).

Make sure to remember the values that you used for the resource group name and Media Services account name.

Create a message endpoint

Before subscribing to the events for the Media Services account, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. In this article, you deploy a [pre-built web app](#) that displays the event messages. The deployed solution includes an App Service plan, an App Service web app, and source code from GitHub.

1. Select **Deploy to Azure** to deploy the solution to your subscription. In the Azure portal, provide values for the parameters.



2. The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: `https://<your-site-name>.azurewebsites.net`

If you switch to the "Azure Event Grid Viewer" site, you see it has no events yet.

Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider.

In the Azure portal:

1. Select **Subscriptions**.
2. Select the subscription you're using for Event Grid.
3. Under **Settings**, select **Resource providers**.
4. Find **Microsoft.EventGrid**.

5. If not registered, select **Register**.

It may take a moment for the registration to finish. Select **Refresh** to update the status. When **Status** is **Registered**, you're ready to continue.

Set the Azure subscription

In the following command, provide the Azure subscription ID that you want to use for the Media Services account. You can see a list of subscriptions that you have access to by navigating to [Subscriptions](#).

```
az account set --subscription mySubscriptionId
```

Subscribe to Media Services events

You subscribe to an article to tell Event Grid which events you want to track. The following example subscribes to the Media Services account you created, and passes the URL from the website you created as the endpoint for event notification.

Replace `<event_subscription_name>` with a unique name for your event subscription. For `<resource_group_name>` and `<ams_account_name>`, use the values you used when creating the Media Services account. For the `<endpoint_URL>`, provide the URL of your web app and add `api/updates` to the home page URL. By specifying the endpoint when subscribing, Event Grid handles the routing of events to that endpoint.

1. Get the resource id

```
amsResourceId=$(az ams account show --name <ams_account_name> --resource-group <resource_group_name> -  
-query id --output tsv)
```

For example:

```
amsResourceId=$(az ams account show --name amsaccount --resource-group amsResourceGroup --query id --  
output tsv)
```

2. Subscribe to the events

```
az eventgrid event-subscription create \  
--resource-id $amsResourceId \  
--name <event_subscription_name> \  
--endpoint <endpoint_URL>
```

For example:

```
az eventgrid event-subscription create --resource-id $amsResourceId --name amsTestEventSubscription --  
endpoint https://amstesteventgrid.azurewebsites.net/api/updates/
```

TIP

You might get validation handshake warning. Give it a few minutes and the handshake should validate.

Now, let's trigger events to see how Event Grid distributes the message to your endpoint.


Send an event to your endpoint

You can trigger events for the Media Services account by running an encoding job. You can follow [this quickstart](#) to encode a file and start sending events.













View your web app again, and notice that a subscription validation event has been sent to it. Event Grid sends the validation event so the endpoint can verify that it wants to receive event data. The endpoint has to set `validationResponse` to `validationCode`. For more information, see [Event Grid security and authentication](#). You can view the web app code to see how it validates the subscription.

TIP

Select the eye icon to expand the event data. Do not refresh the page, if you want to view all the events.



Azure Event Grid Viewer

Event Type	Subject
 Microsoft.Media.JobFinished	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputFinished	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobProcessing	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputProcessing	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputScheduled	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobScheduled	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18
 Microsoft.Media.JobOutputStateChange	transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18

```
{
  "topic": "/subscriptions/ 00000000-0000-0000-000000000000/resourceGroups/amsResourceGroup/providers/Microsoft.Media/mediaservices/iaservices/",
  "subject": "transforms/MyTransformWithAdaptiveStreamingPreset/jobs/job-8bad7d4c6ff843069f6db1e66cbce18c",
  "eventType": "Microsoft.Media.JobOutputStateChange",
  "eventTime": "2018-10-13T08:36:56.264887",
  "id": "4c617372-baec-4f88-8352-65d83d2d3905",
  "data": {
    "previousState": "Queued",
    "output": {
      "@odata.type": "#Microsoft.Media.JobOutputAsset",
      "assetName": "output-8bad7d4c6ff843069f6db1e66cbce18c",
      "error": null,
      "label": "BuiltInStandardEncoderPreset_0",
      "progress": 0,
      "state": "Scheduled"
    },
    "jobCorrelationData": {}
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}
```

Next steps

[Upload, encode, and stream](#)

Use AES-128 dynamic encryption and the key delivery service

10/17/2018 • 11 minutes to read • [Edit Online](#)

You can use Media Services to deliver HTTP Live Streaming (HLS), MPEG-DASH, and Smooth Streaming encrypted with the AES by using 128-bit encryption keys. Media Services also provides the key delivery service that delivers encryption keys to authorized users. If you want for Media Services to encrypt an asset, you associate the encryption key with StreamingLocator and also configure the content key policy. When a stream is requested by a player, Media Services uses the specified key to dynamically encrypt your content by using AES encryption. To decrypt the stream, the player requests the key from the key delivery service. To determine whether the user is authorized to get the key, the service evaluates the content key policy that you specified for the key.

The article is based on the [EncryptWithAES](#) sample. The sample demonstrates how to create an encoding Transform that uses a built-in preset for adaptive bitrate encoding and ingests a file directly from an [HTTPs source URL](#). The output asset is then published using the AES (ClearKey) encryption. The output from the sample is a URL to the Azure Media Player, including both the DASH manifest and the AES token needed to play back the content. The sample sets the expiration of the JWT token to 1 hour. You can open a browser and paste the resulting URL to launch the Azure Media Player demo page with the URL and token filled out for you already in the following format:

```
https://ampdemo.azureedge.net/?url= {dash Manifest URL} &aes=true&aestoken=Bearer%3D{ JWT Token here} .
```

NOTE

You can encrypt each asset with multiple encryption types (AES-128, PlayReady, Widevine, FairPlay). See [Streaming protocols and encryption types](#), to see what makes sense to combine.

Prerequisites

The following are required to complete the tutorial.

- Review the [Content protection overview](#) article.
- Install Visual Studio Code or Visual Studio
- Create a new Azure Media Services account, as described in [this quickstart](#).
- Get credentials needed to use Media Services APIs by following [Access APIs](#)

Download code

Clone a GitHub repository that contains the full .NET sample discussed in this article to your machine using the following command:

```
git clone https://github.com/Azure-Samples/media-services-v3-dotnet-tutorials.git
```

The "Encrypt with AES-128" sample is located in the [EncryptWithAES](#) folder.

NOTE

The sample creates unique resources every time you run the application. Typically, you will reuse existing resources like transforms and policies (if existing resource have required configurations).

Start using Media Services APIs with .NET SDK

To start using Media Services APIs with .NET, you need to create an **AzureMediaServicesClient** object. To create the object, you need to supply credentials needed for the client to connect to Azure using Azure AD. In the code you cloned at the beginning of the article, the **GetCredentialsAsync** function creates the **ServiceClientCredentials** object based on the credentials supplied in the local configuration file.

```
private static async Task<IAzureMediaServicesClient> CreateMediaServicesClientAsync(ConfigWrapper config)
{
    var credentials = await GetCredentialsAsync(config);

    return new AzureMediaServicesClient(config.ArmEndpoint, credentials)
    {
        SubscriptionId = config.SubscriptionId,
    };
}
```

Create an output asset

The output [Asset](#) stores the result of your encoding job.

```
private static async Task<Asset> CreateOutputAssetAsync(IAzureMediaServicesClient client, string
resourceGroupName, string accountName, string assetName)
{
    // Check if an Asset already exists
    Asset outputAsset = await client.Assets.GetAsync(resourceGroupName, accountName, assetName);
    Asset asset = new Asset();
    string outputAssetName = assetName;

    if (outputAsset != null)
    {
        // Name collision! In order to get the sample to work, let's just go ahead and create a unique asset
        name
        // Note that the returned Asset can have a different name than the one specified as an input
        parameter.
        // You may want to update this part to throw an Exception instead, and handle name collisions
        differently.
        string uniqueness = $"-{Guid.NewGuid().ToString("N")}";
        outputAssetName += uniqueness;

        Console.WriteLine("Warning - found an existing Asset with name = " + assetName);
        Console.WriteLine("Creating an Asset with this name instead: " + outputAssetName);
    }

    return await client.Assets.CreateOrUpdateAsync(resourceGroupName, accountName, outputAssetName, asset);
}
```

Get or create an encoding Transform

When creating a new [Transform](#) instance, you need to specify what you want it to produce as an output. The required parameter is a **TransformOutput** object, as shown in the code below. Each **TransformOutput** contains a **Preset**. **Preset** describes the step-by-step instructions of video and/or audio processing operations that are to be used to generate the desired **TransformOutput**. The sample described in this article uses a built-in **Preset**

called **AdaptiveStreaming**. The Preset encodes the input video into an auto-generated bitrate ladder (bitrate-resolution pairs) based on the input resolution and bitrate, and produces ISO MP4 files with H.264 video and AAC audio corresponding to each bitrate-resolution pair.

Before creating a new [Transform](#), you should first check if one already exists using the **Get** method, as shown in the code that follows. In Media Services v3, **Get** methods on entities return **null** if the entity doesn't exist (a case-insensitive check on the name).

```
private static async Task<Transform> GetOrCreateTransformAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName)
{
    // Does a Transform already exist with the desired name? Assume that an existing Transform with the
    // desired name
    // also uses the same recipe or Preset for processing content.
    Transform transform = await client.Transforms.GetAsync(resourceGroupName, accountName, transformName);

    if (transform == null)
    {
        // You need to specify what you want it to produce as an output
        TransformOutput[] output = new TransformOutput[]
        {
            new TransformOutput
            {
                // The preset for the Transform is set to one of Media Services built-in sample presets.
                // You can customize the encoding settings by changing this to use "StandardEncoderPreset"
                class.
                Preset = new BuiltInStandardEncoderPreset()
                {
                    // This sample uses the built-in encoding preset for Adaptive Bitrate Streaming.
                    PresetName = EncoderNamedPreset.AdaptiveStreaming
                }
            }
        };

        // Create the Transform with the output defined above
        transform = await client.Transforms.CreateOrUpdateAsync(resourceGroupName, accountName, transformName,
            output);
    }

    return transform;
}
```

Submit Job

As mentioned above, the [Transform](#) object is the recipe and a [Job](#) is the actual request to Media Services to apply that **Transform** to a given input video or audio content. The **Job** specifies information like the location of the input video, and the location for the output.

In this tutorial, we create the job's input based on a file that is ingested directly from an [HTTPs source URL](#).

```

private static async Task<Job> SubmitJobAsync(IAzureMediaServicesClient client,
    string resourceGroup,
    string accountName,
    string transformName,
    string outputAssetName,
    string jobName)
{
    // This example shows how to encode from any HTTPs source URL - a new feature of the v3 API.
    // Change the URL to any accessible HTTPs URL or SAS URL from Azure.
    JobInputHttp jobInput =
        new JobInputHttp(files: new[] { "https://nimbuscdn-
nimbuspm.streaming.mediaservices.windows.net/2b533311-b215-4409-80af-529c3e853622/ignite-short.mp4" });

    JobOutput[] jobOutputs =
    {
        new JobOutputAsset(outputAssetName),
    };

    // In this example, we are assuming that the job name is unique.
    //
    // If you already have a job with the desired name, use the Jobs.Get method
    // to get the existing job. In Media Services v3, the Get method on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).
    Job job = await client.Jobs.CreateAsync(
        resourceGroup,
        accountName,
        transformName,
        jobName,
        new Job
        {
            Input = jobInput,
            Outputs = jobOutputs,
        });

    return job;
}

```

Wait for the Job to complete

The job takes some time to complete and when it does you want to be notified. The code sample below shows how to poll the service for the status of the [Job](#). Polling is not a recommended best practice for production applications because of potential latency. Polling can be throttled if overused on an account. Developers should instead use Event Grid. See [Route events to a custom web endpoint](#).

The **Job** usually goes through the following states: **Scheduled**, **Queued**, **Processing**, **Finished** (the final state). If the job has encountered an error, you get the **Error** state. If the job is in the process of being canceled, you get **Canceling** and **Canceled** when it is done.

```

private static async Task<Job> WaitForJobToFinishAsync(IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string jobName)
{
    const int SleepIntervalMs = 60 * 1000;

    Job job = null;

    do
    {
        job = await client.Jobs.GetAsync(resourceGroupName, accountName, transformName, jobName);

        Console.WriteLine($"Job is '{job.State}'");
        for (int i = 0; i < job.Outputs.Count; i++)
        {
            JobOutput output = job.Outputs[i];
            Console.WriteLine($"JobOutput[{i}] is '{output.State}'");
            if (output.State == JobState.Processing)
            {
                Console.WriteLine($"    Progress: '{output.Progress}'");
            }

            Console.WriteLine();
        }

        if (job.State != JobState.Finished && job.State != JobState.Error && job.State != JobState.Canceled)
        {
            await Task.Delay(SleepIntervalMs);
        }
    }
    while (job.State != JobState.Finished && job.State != JobState.Error && job.State != JobState.Canceled);

    return job;
}

```

Create a ContentKeyPolicy

A content key provides secure access to your Assets. You need to create a **ContentKeyPolicy** that configures how the content key is delivered to end clients. The content key is associated with **StreamingLocator**. Media Services also provides the key delivery service that delivers encryption keys to authorized users.

When a stream is requested by a player, Media Services uses the specified key to dynamically encrypt your content (in this case, by using AES encryption.) To decrypt the stream, the player requests the key from the key delivery service. To determine whether the user is authorized to get the key, the service evaluates the content key policy that you specified for the key.

```

private static async Task<ContentKeyPolicy> GetOrCreateContentKeyPolicyAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string contentKeyPolicyName)
{
    ContentKeyPolicy policy = await client.ContentKeyPolicies.GetAsync(resourceGroupName, accountName,
contentKeyPolicyName);

    if (policy == null)
    {
        ContentKeyPolicySymmetricTokenKey primaryKey = new ContentKeyPolicySymmetricTokenKey(TokenSigningKey);
        List<ContentKeyPolicyRestrictionTokenKey> alternateKeys = null;
        List<ContentKeyPolicyTokenClaim> requiredClaims = new List<ContentKeyPolicyTokenClaim>()
        {
            ContentKeyPolicyTokenClaim.ContentKeyIdentifierClaim
        };

        List<ContentKeyPolicyOption> options = new List<ContentKeyPolicyOption>()
        {
            new ContentKeyPolicyOption(
                new ContentKeyPolicyClearKeyConfiguration(),
                new ContentKeyPolicyTokenRestriction(Issuer, Audience, primaryKey,
                    ContentKeyPolicyRestrictionTokenType.Jwt, alternateKeys, requiredClaims))
        };

        // Since we are randomly generating the signing key each time, make sure to create or update the
        policy each time.
        // Normally you would use a long lived key so you would just check for the policies existence with Get
        instead of
        // ensuring to create or update it each time.
        policy = await client.ContentKeyPolicies.CreateOrUpdateAsync(resourceGroupName, accountName,
contentKeyPolicyName, options);

    }

    return policy;
}

```

Create a StreamingLocator

After the encoding is complete, and the content key policy is set, the next step is to make the video in the output Asset available to clients for playback. You accomplish this in two steps:

1. Create a [StreamingLocator](#)
2. Build the streaming URLs that clients can use.

The process of creating the **StreamingLocator** is called publishing. By default, the **StreamingLocator** is valid immediately after you make the API calls, and lasts until it is deleted, unless you configure the optional start and end times.

When creating a [StreamingLocator](#), you will need to specify the desired **StreamingPolicyName**. In this tutorial, we are using one of the `PredefinedStreamingPolicies`, which tells Azure Media Services how to publish the content for streaming. In this example, the AES Envelope encryption is applied (also known as ClearKey encryption because the key is delivered to the playback client via HTTPS and not a DRM license).

IMPORTANT

When using a custom [StreamingPolicy](#), you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed. Your Media Service account has a quota for the number of StreamingPolicy entries. You should not be creating a new StreamingPolicy for each StreamingLocator.

```
private static async Task<StreamingLocator> CreateStreamingLocatorAsync(
    IAzureMediaServicesClient client,
    string resourceGroup,
    string accountName,
    string assetName,
    string locatorName,
    string contentPolicyName)
{
    StreamingLocator locator = await client.StreamingLocators.CreateAsync(
        resourceGroup,
        accountName,
        locatorName,
        new StreamingLocator
        {
            AssetName = assetName,
            StreamingPolicyName = PredefinedStreamingPolicy.ClearKey,
            DefaultContentKeyPolicyName = contentPolicyName
        });

    return locator;
}
```

Get a test token

In this tutorial, we specify for the content key policy to have a token restriction. The token-restricted policy must be accompanied by a token issued by a security token service (STS). Media Services supports tokens in the [JSON Web Token](#) (JWT) formats and that is what we configure in the sample.

The ContentKeyIdentifierClaim is used in the ContentKeyPolicy, which means that the token presented to the Key Delivery service must have the identifier of the ContentKey in it. In the sample, we don't specify a content key when creating the StreamingLocator, the system creates a random one for us. In order to generate the test token, we must get the ContentKeyId to put in the ContentKeyIdentifierClaim claim.

```

private static string GetTokenAsync(string issuer, string audience, string keyIdentifier, byte[]
tokenVerificationKey)
{
    var tokenSigningKey = new SymmetricSecurityKey(tokenVerificationKey);

    SigningCredentials cred = new SigningCredentials(
        tokenSigningKey,
        // Use the HmacSha256 and not the HmacSha256Signature option, or the token will not work!
        SecurityAlgorithms.HmacSha256,
        SecurityAlgorithms.Sha256Digest);

    Claim[] claims = new Claim[]
    {
        new Claim(ContentKeyPolicyTokenClaim.ContentKeyIdentifierClaim.ClaimType, keyIdentifier)
    };

    JwtSecurityToken token = new JwtSecurityToken(
        issuer: issuer,
        audience: audience,
        claims: claims,
        notBefore: DateTime.Now.AddMinutes(-5),
        expires: DateTime.Now.AddMinutes(60),
        signingCredentials: cred);

    JwtSecurityTokenHandler handler = new JwtSecurityTokenHandler();

    return handler.WriteToken(token);
}

```

Build a DASH streaming URL

Now that the [StreamingLocator](#) has been created, you can get the streaming URLs. To build a URL, you need to concatenate the [StreamingEndpoint](#) host name and the **StreamingLocator** path. In this sample, the *default StreamingEndpoint* is used. When you first create a Media Service account, this *default StreamingEndpoint* will be in a stopped state, so you need to call **Start**.

```

private static async Task<string> GetDASHStreamingUrlAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string locatorName)
{
    const string DefaultStreamingEndpointName = "default";

    string dashPath = "";

    StreamingEndpoint streamingEndpoint = await client.StreamingEndpoints.GetAsync(resourceGroupName,
accountName, DefaultStreamingEndpointName);

    if (streamingEndpoint != null)
    {
        if (streamingEndpoint.ResourceState != StreamingEndpointResourceState.Running)
        {
            await client.StreamingEndpoints.StartAsync(resourceGroupName, accountName,
DefaultStreamingEndpointName);
        }
    }

    ListPathsResponse paths = await client.StreamingLocators.ListPathsAsync(resourceGroupName, accountName,
locatorName);

    foreach (StreamingPath path in paths.StreamingPaths)
    {
        UriBuilder uriBuilder = new UriBuilder();
        uriBuilder.Scheme = "https";
        uriBuilder.Host = streamingEndpoint.HostName;

        // Look for just the DASH path and generate a URL for the Azure Media Player to playback the content
        with the AES token to decrypt.
        // Note that the JWT token is set to expire in 1 hour.
        if (path.StreamingProtocol == StreamingPolicyStreamingProtocol.Dash)
        {
            uriBuilder.Path = path.Paths[0];

            dashPath = uriBuilder.ToString();

        }
    }

    return dashPath;
}

```

Clean up resources in your Media Services account

Generally, you should clean up everything except objects that you are planning to reuse (typically, you will reuse Transforms, and you will persist StreamingLocators, etc.). If you want for your account to be clean after experimenting, you should delete the resources that you do not plan to reuse. For example, the following code deletes Jobs.

```

private static async Task CleanUpAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string contentKeyPolicyName)
{
    var jobs = await client.Jobs.ListAsync(resourceGroupName, accountName, transformName);
    foreach (var job in jobs)
    {
        await client.Jobs.DeleteAsync(resourceGroupName, accountName, transformName, job.Name);
    }

    var assets = await client.Assets.ListAsync(resourceGroupName, accountName);
    foreach (var asset in assets)
    {
        await client.Assets.DeleteAsync(resourceGroupName, accountName, asset.Name);
    }

    client.ContentKeyPolicies.Delete(resourceGroupName, accountName, contentKeyPolicyName);
}

```

Next steps

Check out how to [protect with DRM](#)

Use DRM dynamic encryption and license delivery service

12/10/2018 • 13 minutes to read • [Edit Online](#)

You can use Azure Media Services to deliver MPEG-DASH, Smooth Streaming, and HTTP Live Streaming (HLS) streams protected with [PlayReady digital rights management \(DRM\)](#). You can also use Media Services to deliver encrypted DASH streams with **Google Widevine** DRM licenses. Both PlayReady and Widevine are encrypted per the common encryption (ISO/IEC 23001-7 CENC) specification. Media Services also enables you to encrypt your HLS content with **Apple FairPlay** (AES-128 CBC).

Furthermore, Media Services provides a service for delivering PlayReady, Widevine, and FairPlay DRM licenses. When a user requests DRM-protected content, the player application requests a license from the Media Services license service. If the player application is authorized, the Media Services license service issues a license to the player. A license contains the decryption key that can be used by the client player to decrypt and stream the content.

This article is based on the [Encrypting with DRM](#) sample. Among other things, the sample demonstrates how to:

- Create an encoding Transform that uses a built-in preset for adaptive bitrate encoding and ingest a file directly from an [HTTPs source URL](#).
- Set the signing key used for verification of your token.
- Set the requirements (restrictions) on the content key policy that must be met to deliver keys with the specified configuration.

- Configuration

In this sample, the [PlayReady](#) and [Widevine](#) licenses are configured so they can be delivered by the Media Services license delivery service. Even though, this sample app does not configure the [FairPlay](#) license, it contains a method that you can use to configure FairPlay. If you wish, you can add FairPlay configuration as another option.

- Restriction

The app sets a JWT token type restriction on the policy.

- Create a StreamingLocator for the specified asset and with the specified streaming policy name. In this case, the predefined policy is used. It sets two content keys on the StreamingLocator: AES-128 (envelope) and CENC (PlayReady and Widevine).

Once the StreamingLocator is created the output asset is published and available to clients for playback.

NOTE

Make sure the StreamingEndpoint from which you want to stream is in the Running state.

- Create a URL, to the Azure Media Player, that includes both the DASH manifest and the PlayReady token needed to play back the PlayReady encrypted content. The sample sets the expiration of the token to 1 hour.

You can open a browser and paste the resulting URL to launch the Azure Media Player demo page with the URL and token filled out for you already.

```
C:\WINDOWS\system32\cmd.exe
Job is 'Queued'.
JobOutput[0] is 'Queued'.
Job is 'Processing'.
JobOutput[0] is 'Processing'. Progress: '0'.
Job is 'Processing'.
JobOutput[0] is 'Processing'. Progress: '31'.
Job is 'Processing'.
JobOutput[0] is 'Processing'. Progress: '63'.
Job is 'Finished'.
JobOutput[0] is 'Finished'.
KeyIdentifier = 51cf92a4-ae42-4673-b7c1-4ab4f5f2e267
Copy and paste the following URL in your browser to play back the file in the Azure Media Player.


https://ampdemo.azureedge.net/?url=https://juliakoamsaccount-usw22.streaming.media.azure.net/dd2dfcf9-2b74-4cf8-b0cb-6eb0668ca2e0/ignite-short.ism/manifest(format=mpd-time-csf,encryption=cenc)&playready=true&playreadytoken=Bearer%3DeyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1cm46bWljcm9zb2Z0OmF6dXJlOm1lZGhlc2Vydm1jZXM6Y29udGVudGtleWlkZW50aWZpZSI6IjE1MmNmOTJhNC1hZTQyLTQ2NzhtYjZjMS00YWI0ZjVmMmUyNjc1Ij0uYmYiOiJlMzE3MTA5OTAsImV4cCI6MTUzMTcxNDg5MCMwIiwiaXNzIjoibXJlc3N1ZDIiLCJhdWQiOiJ1teUF1ZGllbmNlIn0.kZ2vab_VZ_sFCsaqbpLo8tHigYptf6Z0F5Z6RoERIq4

When finished testing press enter to cleanup.
```

NOTE

You can encrypt each asset with multiple encryption types (AES-128, PlayReady, Widevine, FairPlay). See [Streaming protocols and encryption types](#), to see what makes sense to combine.

The sample described in this article produces the following result:

 Azure Media Player

Microsoft

This demo page allows you to try out some of the features of Azure Media Player.

1. Select a sample or input a video URL from your Media Services account
2. Update Player

Copy the URL to share this page.

get share url

Don't have an Azure Media Services account? No worries:



start your free trial

Chosen Player Options

Format: DASH
Tech: JavaScript + HTML5 (MSE)
Protection: PlayReady
AMP Version: 2.1.7

▶

0:04 / 1:10



Setup

Diagnostics

Code

Adaptive streaming URL from Azure Media Services

Samples:

Select a sample

URL:

https://juliakoamsaccount-usw22.streaming.media.azure.net/dd2dfcf9-2b74-4cf8-b0cb-6eb0668ca2e0/ignite-short.ism/manifest(format=mpd-time-csf,encryption=cenc)&playready=true&playreadytoken=Bearer%3DeyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1cm46bWljcm9zb2Z0OmF6dXJlOm1lZGhlc2Vydm1jZXM6Y29udGVudGtleWlkZW50aWZpZSI6IjE1MmNmOTJhNC1hZTQyLTQ2NzhtYjZjMS00YWI0ZjVmMmUyNjc1Ij0uYmYiOiJlMzE3MTA5OTAsImV4cCI6MTUzMTcxNDg5MCMwIiwiaXNzIjoibXJlc3N1ZDIiLCJhdWQiOiJ1teUF1ZGllbmNlIn0.kZ2vab_VZ_sFCsaqbpLo8tHigYptf6Z0F5Z6RoERIq4

Advanced Options

Please note: not all formats are playable by all techs

Playback:

Hybrid Heuristic Profile

☒ AutoPlay (if applicable)

Format:

Azure Auto-Generated

☐ Disable Automatic URL Rewriter

Tech:

Auto Select

☐ Muted (on start)

Language:

English

Protection: ☐ AES

AES Token

☒ PlayReady

Bearer=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1cm46bWljcm9zb2Z0OmF6dXJlOm1lZGhlc2Vydm1jZXM6Y29udGVudGtleWlkZW50aWZpZSI6IjE1MmNmOTJhNC1hZTQyLTQ2NzhtYjZjMS00YWI0ZjVmMmUyNjc1Ij0uYmYiOiJlMzE3MTA5OTAsImV4cCI6MTUzMTcxNDg5MCMwIiwiaXNzIjoibXJlc3N1ZDIiLCJhdWQiOiJ1teUF1ZGllbmNlIn0.kZ2vab_VZ_sFCsaqbpLo8tHigYptf6Z0F5Z6RoERIq4

☐ Widevine

Widevine Token

Prerequisites

The following are required to complete the tutorial.

- Review the [Content protection overview](#) article.
- Review the [Design multi-drm content protection system with access control](#)
- Install Visual Studio Code or Visual Studio
- Create a new Azure Media Services account, as described in [this quickstart](#).
- Get credentials needed to use Media Services APIs by following [Access APIs](#)
- Set the appropriate values in the application configuration file (appsettings.json).

Download code

Clone a GitHub repository that contains the full .NET sample discussed in this article to your machine using the following command:

```
git clone https://github.com/Azure-Samples/media-services-v3-dotnet-tutorials.git
```

The "Encrypt with DRM" sample is located in the [EncryptWithDRM](#) folder.

NOTE

The sample creates unique resources every time you run the application. Typically, you will reuse existing resources like transforms and policies (if existing resource have required configurations).

Start using Media Services APIs with .NET SDK

To start using Media Services APIs with .NET, you need to create an **AzureMediaServicesClient** object. To create the object, you need to supply credentials needed for the client to connect to Azure using Azure AD. In the code you cloned at the beginning of the article, the **GetCredentialsAsync** function creates the **ServiceClientCredentials** object based on the credentials supplied in the local configuration file.

```
private static async Task<IAzureMediaServicesClient> CreateMediaServicesClientAsync(ConfigWrapper config)
{
    var credentials = await GetCredentialsAsync(config);

    return new AzureMediaServicesClient(config.ArmEndpoint, credentials)
    {
        SubscriptionId = config.SubscriptionId,
    };
}
```

Create an output asset

The output [Asset](#) stores the result of your encoding job.

```

private static async Task<Asset> CreateOutputAssetAsync(IAzureMediaServicesClient client, string
resourceGroupName, string accountName, string assetName)
{
    // Check if an Asset already exists
    Asset outputAsset = await client.Assets.GetAsync(resourceGroupName, accountName, assetName);
    Asset asset = new Asset();
    string outputAssetName = assetName;

    if (outputAsset != null)
    {
        // Name collision! In order to get the sample to work, let's just go ahead and create a unique asset
name
        // Note that the returned Asset can have a different name than the one specified as an input
parameter.
        // You may want to update this part to throw an Exception instead, and handle name collisions
differently.
        string uniqueness = $"-{Guid.NewGuid().ToString("N")}";
        outputAssetName += uniqueness;

        Console.WriteLine("Warning - found an existing Asset with name = " + assetName);
        Console.WriteLine("Creating an Asset with this name instead: " + outputAssetName);
    }

    return await client.Assets.CreateOrUpdateAsync(resourceGroupName, accountName, outputAssetName, asset);
}

```

Get or create an encoding Transform

When creating a new [Transform](#) instance, you need to specify what you want it to produce as an output. The required parameter is a **TransformOutput** object, as shown in the code below. Each **TransformOutput** contains a **Preset**. **Preset** describes the step-by-step instructions of video and/or audio processing operations that are to be used to generate the desired **TransformOutput**. The sample described in this article uses a built-in Preset called **AdaptiveStreaming**. The Preset encodes the input video into an auto-generated bitrate ladder (bitrate-resolution pairs) based on the input resolution and bitrate, and produces ISO MP4 files with H.264 video and AAC audio corresponding to each bitrate-resolution pair.

Before creating a new [Transform](#), you should first check if one already exists using the **Get** method, as shown in the code that follows. In Media Services v3, **Get** methods on entities return **null** if the entity doesn't exist (a case-insensitive check on the name).


```

private static async Task<Transform> GetOrCreateTransformAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName)
{
    // Does a Transform already exist with the desired name? Assume that an existing Transform with the
    // desired name
    // also uses the same recipe or Preset for processing content.
    Transform transform = await client.Transforms.GetAsync(resourceGroupName, accountName, transformName);

    if (transform == null)
    {
        // You need to specify what you want it to produce as an output
        TransformOutput[] output = new TransformOutput[]
        {
            new TransformOutput
            {
                // The preset for the Transform is set to one of Media Services built-in sample presets.
                // You can customize the encoding settings by changing this to use "StandardEncoderPreset"
                class.
                Preset = new BuiltInStandardEncoderPreset()
                {
                    // This sample uses the built-in encoding preset for Adaptive Bitrate Streaming.
                    PresetName = EncoderNamedPreset.AdaptiveStreaming
                }
            }
        };

        // Create the Transform with the output defined above
        transform = await client.Transforms.CreateOrUpdateAsync(resourceGroupName, accountName,
            transformName, output);
    }

    return transform;
}

```

Submit Job

As mentioned above, the **Transform** object is the recipe and a **Job** is the actual request to Media Services to apply that **Transform** to a given input video or audio content. The **Job** specifies information like the location of the input video, and the location for the output.

In this tutorial, we create the job's input based on a file that is ingested directly from an [HTTPs source URL](#).

```

private static async Task<Job> SubmitJobAsync(IAzureMediaServicesClient client,
    string resourceGroup,
    string accountName,
    string transformName,
    string outputAssetName,
    string jobName)
{
    // This example shows how to encode from any HTTPs source URL - a new feature of the v3 API.
    // Change the URL to any accessible HTTPs URL or SAS URL from Azure.
    JobInputHttp jobInput =
        new JobInputHttp(files: new[] { "https://nimbuscdn-
nimbuspm.streaming.mediaservices.windows.net/2b533311-b215-4409-80af-529c3e853622/ignite-short.mp4" });

    JobOutput[] jobOutputs =
    {
        new JobOutputAsset(outputAssetName),
    };

    // In this example, we are assuming that the job name is unique.
    //
    // If you already have a job with the desired name, use the Jobs.Get method
    // to get the existing job. In Media Services v3, the Get method on entities returns null
    // if the entity doesn't exist (a case-insensitive check on the name).
    Job job = await client.Jobs.CreateAsync(
        resourceGroup,
        accountName,
        transformName,
        jobName,
        new Job
        {
            Input = jobInput,
            Outputs = jobOutputs,
        });

    return job;
}

```

Wait for the Job to complete

The job takes some time to complete and when it does you want to be notified. The code sample below shows how to poll the service for the status of the [Job](#). Polling is not a recommended best practice for production applications because of potential latency. Polling can be throttled if overused on an account. Developers should instead use Event Grid. See [Route events to a custom web endpoint](#).

The **Job** usually goes through the following states: **Scheduled**, **Queued**, **Processing**, **Finished** (the final state). If the job has encountered an error, you get the **Error** state. If the job is in the process of being canceled, you get **Canceling** and **Canceled** when it is done.

```

private static async Task<Job> WaitForJobToFinishAsync(IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string jobName)
{
    const int SleepIntervalMs = 60 * 1000;

    Job job = null;

    do
    {
        job = await client.Jobs.GetAsync(resourceGroupName, accountName, transformName, jobName);

        Console.WriteLine($"Job is '{job.State}'");
        for (int i = 0; i < job.Outputs.Count; i++)
        {
            JobOutput output = job.Outputs[i];
            Console.WriteLine($"JobOutput[{i}] is '{output.State}'");
            if (output.State == JobState.Processing)
            {
                Console.WriteLine($"Progress: '{output.Progress}'");
            }

            Console.WriteLine();
        }

        if (job.State != JobState.Finished && job.State != JobState.Error && job.State != JobState.Canceled)
        {
            await Task.Delay(SleepIntervalMs);
        }
    }
    while (job.State != JobState.Finished && job.State != JobState.Error && job.State != JobState.Canceled);

    return job;
}

```

Create a ContentKeyPolicy

A content key provides secure access to your Assets. You need to create a content key policy that configures how the content key is delivered to end clients. The content key is associated with StreamingLocator. Media Services also provides the key delivery service that delivers encryption keys and licenses to authorized users.

You need to set the requirements (restrictions) on the content key policy that must be met to deliver keys with the specified configuration. In this example, we set the following configurations and requirements:

- Configuration

The [PlayReady](#) and [Widevine](#) licenses are configured so they can be delivered by the Media Services license delivery service. Even though, this sample app does not configure the [FairPlay](#) license, it contains a method that you can use to configure FairPlay. You can add FairPlay configuration as another option.

- Restriction

The app sets a JWT token type restriction on the policy.

When a stream is requested by a player, Media Services uses the specified key to dynamically encrypt your content. To decrypt the stream, the player requests the key from the key delivery service. To determine whether the user is authorized to get the key, the service evaluates the content key policy that you specified for the key.

```

private static async Task<ContentKeyPolicy> GetOrCreateContentKeyPolicyAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string contentKeyPolicyName,
    byte[] tokenSigningKey)
{
    ContentKeyPolicy policy = await client.ContentKeyPolicies.GetAsync(resourceGroupName, accountName,
contentKeyPolicyName);

    if (policy == null)
    {
        ContentKeyPolicySymmetricTokenKey primaryKey = new
ContentKeyPolicySymmetricTokenKey(tokenSigningKey);
        List<ContentKeyPolicyTokenClaim> requiredClaims = new List<ContentKeyPolicyTokenClaim>()
        {
            ContentKeyPolicyTokenClaim.ContentKeyIdentifierClaim
        };
        List<ContentKeyPolicyRestrictionTokenKey> alternateKeys = null;
        ContentKeyPolicyTokenRestriction restriction
            = new ContentKeyPolicyTokenRestriction(Issuer, Audience, primaryKey,
ContentKeyPolicyRestrictionTokenType.Jwt, alternateKeys, requiredClaims);

        ContentKeyPolicyPlayReadyConfiguration playReadyConfig = ConfigurePlayReadyLicenseTemplate();
        ContentKeyPolicyWidevineConfiguration widevineConfig = ConfigureWidevineLicenseTempate();

        List<ContentKeyPolicyOption> options = new List<ContentKeyPolicyOption>();

        options.Add(
            new ContentKeyPolicyOption()
            {
                Configuration = playReadyConfig,
                // If you want to set an open restriction, use
                // Restriction = new ContentKeyPolicyOpenRestriction()
                Restriction = restriction
            });

        options.Add(
            new ContentKeyPolicyOption()
            {
                Configuration = widevineConfig,
                Restriction = restriction
            });

        policy = await client.ContentKeyPolicies.CreateOrUpdateAsync(resourceGroupName, accountName,
contentKeyPolicyName, options);
    }
    else
    {
        // Get the signing key from the existing policy.
        var policyProperties = await
client.ContentKeyPolicies.GetPolicyPropertiesWithSecretsAsync(resourceGroupName, accountName,
contentKeyPolicyName);
        var restriction = policyProperties.Options[0].Restriction as ContentKeyPolicyTokenRestriction;
        if (restriction != null)
        {
            var signingKey = restriction.PrimaryVerificationKey as ContentKeyPolicySymmetricTokenKey;
            if (signingKey != null)
            {
                TokenSigningKey = signingKey.KeyValue;
            }
        }
    }
    return policy;
}

```

Create a StreamingLocator

After the encoding is complete, and the content key policy is set, the next step is to make the video in the output Asset available to clients for playback. You accomplish this in two steps:

1. Create a [StreamingLocator](#)
2. Build the streaming URLs that clients can use.

The process of creating the **StreamingLocator** is called publishing. By default, the **StreamingLocator** is valid immediately after you make the API calls, and lasts until it is deleted, unless you configure the optional start and end times.

When creating a [StreamingLocator](#), you need to specify the desired **StreamingPolicyName**. In this tutorial, we are using one of the predefined StreamingPolicies, which tells Azure Media Services how to publish the content for streaming. In this example, we set StreamingLocator.StreamingPolicyName to the "Predefined_MultiDrmCencStreaming" policy. This policy indicates that you want for two content keys (envelope and CENC) to get generated and set on the locator. Thus, the envelope, PlayReady, and Widevine encryptions are applied (the key is delivered to the playback client based on the configured DRM licenses). If you also want to encrypt your stream with CBCS (FairPlay), use "Predefined_MultiDrmStreaming".

IMPORTANT

When using a custom [StreamingPolicy](#), you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed. Your Media Service account has a quota for the number of StreamingPolicy entries. You should not be creating a new StreamingPolicy for each StreamingLocator.

```
private static async Task<StreamingLocator> CreateStreamingLocatorAsync(
    IAzureMediaServicesClient client,
    string resourceGroup,
    string accountName,
    string assetName,
    string locatorName,
    string contentPolicyName)
{
    // If you also added FairPlay, use "Predefined_MultiDrmStreaming"
    StreamingLocator locator = await client.StreamingLocators.CreateAsync(
        resourceGroup,
        accountName,
        locatorName,
        new StreamingLocator
        {
            AssetName = assetName,
            // "Predefined_MultiDrmCencStreaming" policy supports envelope and cenc encryption
            // And sets two content keys on the StreamingLocator
            StreamingPolicyName = "Predefined_MultiDrmCencStreaming",
            DefaultContentKeyPolicyName = contentPolicyName
        });

    return locator;
}
```

Get a test token

In this tutorial, we specify for the content key policy to have a token restriction. The token-restricted policy must be accompanied by a token issued by a security token service (STS). Media Services supports tokens in the [JSON Web Token](#) (JWT) formats and that is what we configure in the sample.

The `ContentKeyIdentifierClaim` is used in the `ContentKeyPolicy`, which means that the token presented to the key delivery service must have the identifier of the `ContentKey` in it. In the sample, we don't specify a content key when creating the `StreamingLocator`, the system creates a random one for us. In order to generate the test token, we must get the `ContentKeyId` to put in the `ContentKeyIdentifierClaim` claim.

```
private static string GetTokenAsync(string issuer, string audience, string keyIdentifier, byte[]
tokenVerificationKey)
{
    var tokenSigningKey = new SymmetricSecurityKey(tokenVerificationKey);

    SigningCredentials cred = new SigningCredentials(
        tokenSigningKey,
        // Use the HmacSha256 and not the HmacSha256Signature option, or the token will not work!
        SecurityAlgorithms.HmacSha256,
        SecurityAlgorithms.Sha256Digest);

    Claim[] claims = new Claim[]
    {
        new Claim(ContentKeyPolicyTokenClaim.ContentKeyIdentifierClaim.ClaimType, keyIdentifier)
    };

    JwtSecurityToken token = new JwtSecurityToken(
        issuer: issuer,
        audience: audience,
        claims: claims,
        notBefore: DateTime.Now.AddMinutes(-5),
        expires: DateTime.Now.AddMinutes(60),
        signingCredentials: cred);

    JwtSecurityTokenHandler handler = new JwtSecurityTokenHandler();

    return handler.WriteToken(token);
}
```

Build a DASH streaming URL

Now that the [StreamingLocator](#) has been created, you can get the streaming URLs. To build a URL, you need to concatenate the [StreamingEndpoint](#) host name and the **StreamingLocator** path. In this sample, the *default StreamingEndpoint* is used. When you first create a Media Service account, this *default StreamingEndpoint* will be in a stopped state, so you need to call **Start**.

```

private static async Task<string> GetDASHStreamingUrlAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string locatorName)
{
    const string DefaultStreamingEndpointName = "default";

    string dashPath = "";

    StreamingEndpoint streamingEndpoint = await client.StreamingEndpoints.GetAsync(resourceGroupName,
accountName, DefaultStreamingEndpointName);

    if (streamingEndpoint != null)
    {
        if (streamingEndpoint.ResourceState != StreamingEndpointResourceState.Running)
        {
            await client.StreamingEndpoints.StartAsync(resourceGroupName, accountName,
DefaultStreamingEndpointName);
        }
    }

    ListPathsResponse paths = await client.StreamingLocators.ListPathsAsync(resourceGroupName, accountName,
locatorName);

    foreach (StreamingPath path in paths.StreamingPaths)
    {
        UriBuilder uriBuilder = new UriBuilder();
        uriBuilder.Scheme = "https";
        uriBuilder.Host = streamingEndpoint.HostName;

        // Look for just the DASH path and generate a URL for the Azure Media Player to playback the
encrypted DASH content.
        // Note that the JWT token is set to expire in 1 hour.
        if (path.StreamingProtocol == StreamingPolicyStreamingProtocol.Dash)
        {
            uriBuilder.Path = path.Paths[0];

            dashPath = uriBuilder.ToString();

        }
    }

    return dashPath;
}

```

Clean up resources in your Media Services account

Generally, you should clean up everything except objects that you are planning to reuse (typically, you will reuse Transforms, and you will persist StreamingLocators, etc.). If you want for your account to be clean after experimenting, you should delete the resources that you do not plan to reuse. For example, the following code deletes Jobs.

```

private static async Task CleanUpAsync(
    IAzureMediaServicesClient client,
    string resourceGroupName,
    string accountName,
    string transformName,
    string contentKeyPolicyName)
{
    var jobs = await client.Jobs.ListAsync(resourceGroupName, accountName, transformName);
    foreach (var job in jobs)
    {
        await client.Jobs.DeleteAsync(resourceGroupName, accountName, transformName, job.Name);
    }

    var streamingLocators = await client.StreamingLocators.ListAsync(resourceGroupName, accountName);
    foreach (var locator in streamingLocators)
    {
        await client.StreamingLocators.DeleteAsync(resourceGroupName, accountName, locator.Name);
    }

    var assets = await client.Assets.ListAsync(resourceGroupName, accountName);
    foreach (var asset in assets)
    {
        await client.Assets.DeleteAsync(resourceGroupName, accountName, asset.Name);
    }

    client.ContentKeyPolicies.Delete(resourceGroupName, accountName, contentKeyPolicyName);
}

```

Next steps

Check out how to [protect with AES-128](#)

Configure Postman for Media Services REST API calls

12/18/2018 • 2 minutes to read • [Edit Online](#)

This article shows you how to configure **Postman** so it can be used to call Azure Media Services (AMS) REST APIs. The article shows how to import environment and collection files into **Postman**. The collection contains grouped definitions of HTTP requests that call Azure Media Services (AMS) REST APIs. The environment file contains variables that are used by the collection.

Prerequisites

- [Create a Media Services account](#). Make sure to remember the resource group name and the Media Services account name.
- Get information needed to [access APIs](#)
- Install the [Postman](#) REST client to execute the REST APIs shown in some of the AMS REST tutorials.

We are using **Postman** but any REST tool would be suitable. Other alternatives are: **Visual Studio Code** with the REST plugin or **Telerik Fiddler**.

Download Postman files

Clone a GitHub repository that contains the Postman collection and environment files.

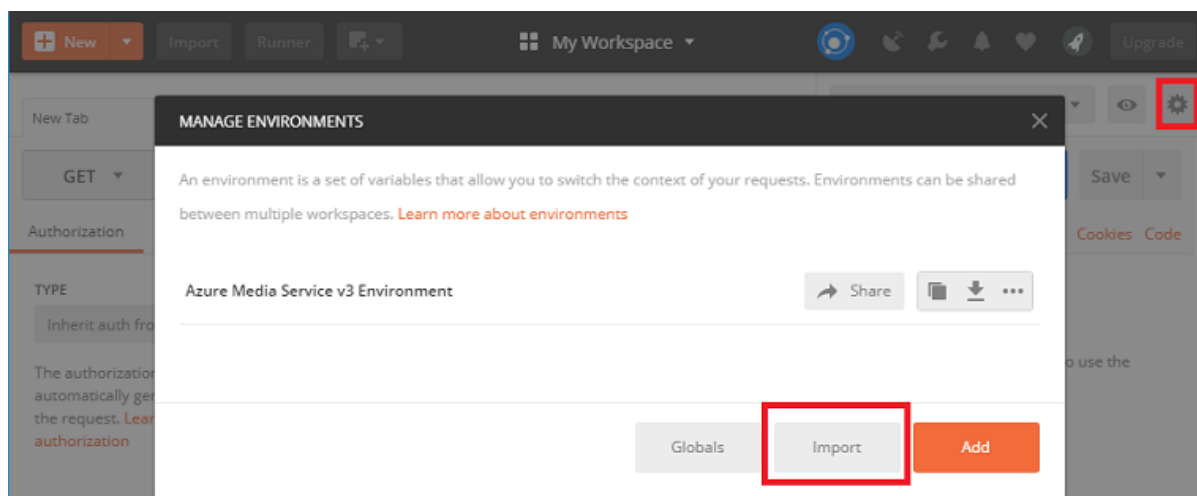
```
git clone https://github.com/Azure-Samples/media-services-v3-rest-postman.git
```

Configure Postman

This section configures the Postman.

Configure the environment

1. Open the **Postman**.
2. On the right of the screen, select the **Manage environment** option.



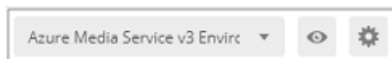
3. From the **Manage environment** dialog, click **Import**.
4. Browse to the `Azure Media Service v3 Environment.postman_environment.json` file that was downloaded when you cloned `https://github.com/Azure-Samples/media-services-v3-rest-postman.git`.

5. The **Azure Media Service v3 Environment** environment is added.

NOTE

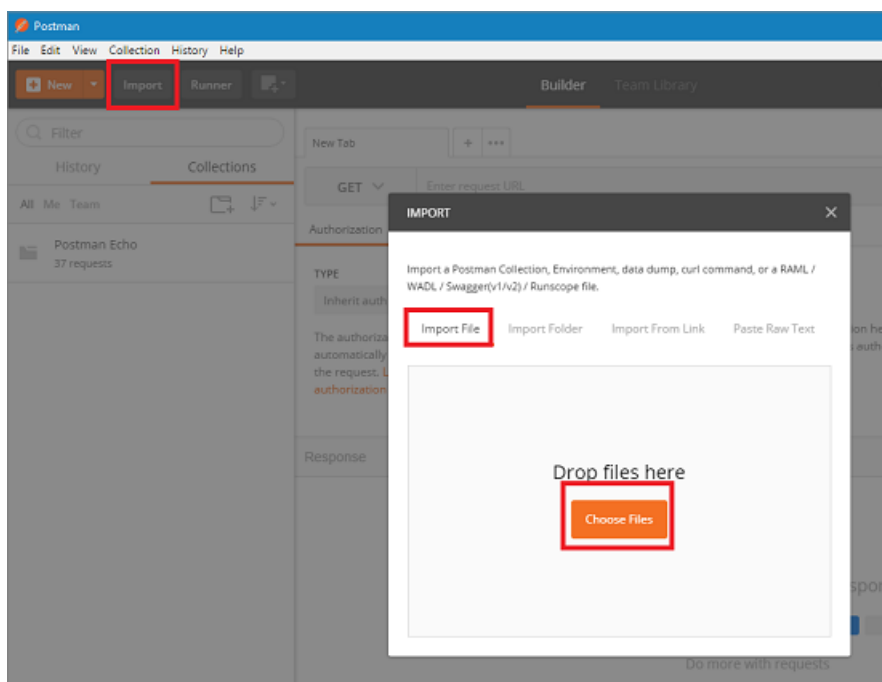
Update access variables with values you got from the **Access the Media Services API** section above.

6. Double-click on the selected file and enter values that you got by following the [accessing API](#) steps.
7. Close the dialog.
8. Select the **Azure Media Service v3 Environment** environment from the dropdown.



Configure the collection

1. Click **Import** to import the collection file.
2. Browse to the `Media Services v3.postman_collection.json` file that was downloaded when you cloned <https://github.com/Azure-Samples/media-services-v3-rest-postman.git>
3. Choose the **Media Services v3.postman_collection.json** file.



Get Azure AD Token

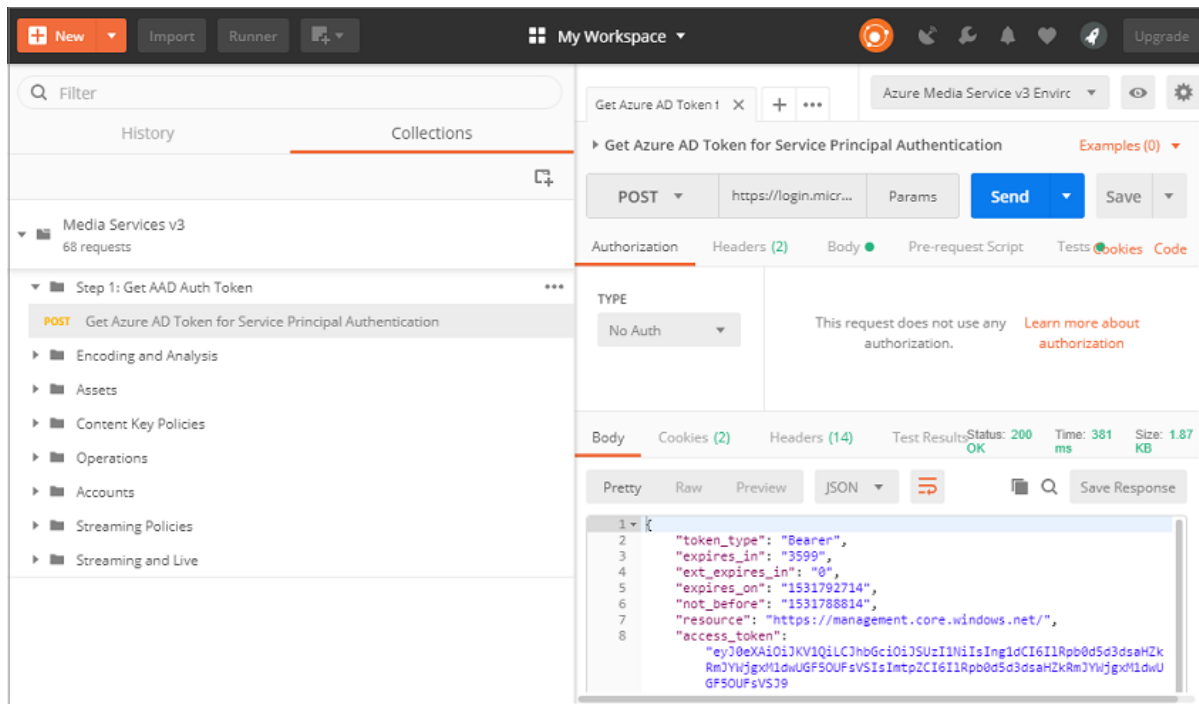
Before you start manipulating AMS v3 resources you need to get and set Azure AD Token for Service Principal Authentication.

1. In the left window of the Postman, select "Step 1: Get AAD Auth token".
2. Then, select "Get Azure AD Token for Service Principal Authentication".
3. Press **Send**.

The following **POST** operation is sent.

```
https://login.microsoftonline.com/:tenantId/oauth2/token
```

4. The response comes back with the token and sets the "AccessToken" environment variable to the token value.



Next steps

Stream files with REST.

Scaling media processing

12/10/2018 • 2 minutes to read • [Edit Online](#)

Azure Media Services enables you to scale media processing in your account by managing Media Reserved Units (MRUs). For detailed overview, see [Scaling media processing](#).

This article shows how to use [Media Services v3 CLI](#) to scale MRUs.

NOTE

For the Audio Analysis and Video Analysis Jobs that are triggered by Media Services v3 or Video Indexer, it is highly recommended to provision your account with 10 S3 MRUs.

If you need more than 10 S3 MRUs, open a support ticket using the [Azure portal](#).

Prerequisites

- Install and use the CLI locally, this article requires the Azure CLI version 2.0 or later. Run `az --version` to find the version you have. If you need to install or upgrade, see [Install the Azure CLI](#).

Currently, not all [Media Services v3 CLI](#) commands work in the Azure Cloud Shell. It is recommended to use the CLI locally.

- [Create a Media Services account](#).

Scale Media Reserved Units with CLI

The following `az ams account mru` command sets Media Reserved Units on the "amsaccount" account using the **count** and **type** parameters.

```
az account set mru -n amsaccount -g amsResourceGroup --count 10 --type S3
```

Billing

You are charged based on the number, type, and amount of time that MRUs are provisioned in your account. Charges apply whether or not you run any jobs. For a detailed explanation, see the FAQ section of the [Media Services pricing](#) page.

Next step

[Analyze videos](#)

See also

[Azure CLI](#)

Azure Event Grid schemas for Media Services events

12/6/2018 • 11 minutes to read • [Edit Online](#)

This article provides the schemas and properties for Media Services events.

For a list of sample scripts and tutorials, see [Media Services event source](#).

Available event types

Job related event types

Media Services emits the **Job** related event types described below. There are two categories for the **Job** related events: "Monitoring Job State Changes" and "Monitoring Job Output State Changes".

You can register for all of the events by subscribing to the JobStateChange event. Or, you can subscribe for specific events only (for example, final states like JobErrored, JobFinished, and JobCanceled).

Monitoring Job State Changes

EVENT TYPE	DESCRIPTION
Microsoft.Media.JobStateChange	Get an event for all Job State changes.
Microsoft.Media.JobScheduled	Get an event when Job transitions to scheduled state.
Microsoft.Media.JobProcessing	Get an event when Job transitions to processing state.
Microsoft.Media.JobCanceling	Get an event when Job transitions to canceling state.
Microsoft.Media.JobFinished	Get an event when Job transitions to finished state. This is a final state that includes Job outputs.
Microsoft.Media.JobCanceled	Get an event when Job transitions to canceled state. This is a final state that includes Job outputs.
Microsoft.Media.JobErrored	Get an event when Job transitions to error state. This is a final state that includes Job outputs.

Monitoring Job Output State Changes

EVENT TYPE	DESCRIPTION
Microsoft.Media.JobOutputStateChange	Get an event for all Job output State changes.
Microsoft.Media.JobOutputScheduled	Get an event when Job output transitions to scheduled state.
Microsoft.Media.JobOutputProcessing	Get an event when Job output transitions to processing state.
Microsoft.Media.JobOutputCanceling	Get an event when Job output transitions to canceling state.
Microsoft.Media.JobOutputFinished	Get an event when Job output transitions to finished state.

EVENT TYPE	DESCRIPTION
Microsoft.Media.JobOutputCanceled	Get an event when Job output transitions to canceled state.
Microsoft.Media.JobOutputErrored	Get an event when Job output transitions to error state.

Live event types

Media Services also emits the **Live** event types described below. There are two categories for the **Live** events: stream-level events and track-level events.

Stream-level events

Stream-level events are raised per stream or connection. Each event has a `StreamId` parameter that identifies the connection or stream. Each stream or connection has one or more tracks of different types. For example, one connection from an encoder may have one audio track and four video tracks. The stream event types are:

EVENT TYPE	DESCRIPTION
Microsoft.Media.LiveEventConnectionRejected	Encoder's connection attempt is rejected.
Microsoft.Media.LiveEventEncoderConnected	Encoder establishes connection with live event.
Microsoft.Media.LiveEventEncoderDisconnected	Encoder disconnects.

Track-level events

Track-level events are raised per track. The track event types are:

EVENT TYPE	DESCRIPTION
Microsoft.Media.LiveEventIncomingDataChunkDropped	Media server drops data chunk because it's too late or has an overlapping timestamp (timestamp of new data chunk is less than the end time of the previous data chunk).
Microsoft.Media.LiveEventIncomingStreamReceived	Media server receives first data chunk for each track in the stream or connection.
Microsoft.Media.LiveEventIncomingStreamsOutOfSync	Media server detects audio and video streams are out of sync. Use as a warning because user experience may not be impacted.
Microsoft.Media.LiveEventIncomingVideoStreamsOutOfSync	Media server detects any of the two video streams coming from external encoder are out of sync. Use as a warning because user experience may not be impacted.
Microsoft.Media.LiveEventIngestHeartbeat	Published every 20 seconds for each track when live event is running. Provides ingest health summary.
Microsoft.Media.LiveEventTrackDiscontinuityDetected	Media server detects discontinuity in the incoming track.

Event schemas and properties

JobStateChange

The following example shows the schema of the **JobStateChange** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
    "eventType": "Microsoft.Media.JobStateChange",
    "eventTime": "2018-04-20T21:26:13.8978772",
    "id": "b9d38923-9210-4c2b-958f-0054467d4dd7",
    "data": {
      "previousState": "Processing",
      "state": "Finished"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
previousState	string	The state of the job before the event.
state	string	The new state of the job being notified in this event. For example, "Scheduled: The job is ready to start" or "Finished: The job is finished" .

Where the Job state can be one of the values: *Queued*, *Scheduled*, *Processing*, *Finished*, *Error*, *Canceled*, *Canceling*

NOTE

Queued is only going to be present in the **previousState** property but not in the **state** property.

JobScheduled, JobProcessing, JobCanceling

For each non-final Job state change (such as JobScheduled, JobProcessing, JobCanceling), the example schema looks similar to the following:

```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
  "eventType": "Microsoft.Media.JobProcessing",
  "eventTime": "2018-10-12T16:12:18.0839935",
  "id": "a0a6efc8-f647-4fc2-be73-861fa25ba2db",
  "data": {
    "previousState": "Scheduled",
    "state": "Processing",
    "correlationData": {
      "TestKey1": "TestValue1",
      "testKey2": "testValue2"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

JobFinished, JobCanceled, JobErrored

For each final Job state change (such as JobFinished, JobCanceled, JobErrored), the example schema looks similar

to the following:

```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
  "eventType": "Microsoft.Media.JobFinished",
  "eventTime": "2018-10-12T16:25:56.4115495",
  "id": "9e07e83a-dd6e-466b-a62f-27521b216f2a",
  "data": {
    "outputs": [
      {
        "@odata.type": "#Microsoft.Media.JobOutputAsset",
        "assetName": "output-7640689F",
        "error": null,
        "label": "VideoAnalyzerPreset_0",
        "progress": 100,
        "state": "Finished"
      }
    ],
    "previousState": "Processing",
    "state": "Finished",
    "correlationData": {
      "TestKey1": "TestValue1",
      "testKey2": "testValue2"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
Outputs	Array	Gets the Job outputs.

JobOutputStateChange

The following example shows the schema of the **JobOutputStateChange** event:


```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-
name>/providers/Microsoft.Media/mediaservices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
  "eventType": "Microsoft.Media.JobOutputStateChange",
  "eventTime": "2018-10-12T16:25:56.0242854",
  "id": "dde85f46-b459-4775-b5c7-befe8e32cf90",
  "data": {
    "previousState": "Processing",
    "output": {
      "@odata.type": "#Microsoft.Media.JobOutputAsset",
      "assetName": "output-7640689F",
      "error": null,
      "label": "VideoAnalyzerPreset_0",
      "progress": 100,
      "state": "Finished"
    },
    "jobCorrelationData": {
      "TestKey1": "TestValue1",
      "testKey2": "testValue2"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

JobOutputScheduled, JobOutputProcessing, JobOutputFinished, JobOutputCanceling, JobOutputCanceled, JobOutputErrored

For each JobOutput state change, the example schema looks similar to the following:

```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-
name>/providers/Microsoft.Media/mediaservices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
  "eventType": "Microsoft.Media.JobOutputProcessing",
  "eventTime": "2018-10-12T16:12:18.0061141",
  "id": "f1fd5338-1b6c-4e31-83c9-cd7c88d2aedb",
  "data": {
    "previousState": "Scheduled",
    "output": {
      "@odata.type": "#Microsoft.Media.JobOutputAsset",
      "assetName": "output-7640689F",
      "error": null,
      "label": "VideoAnalyzerPreset_0",
      "progress": 0,
      "state": "Processing"
    },
    "jobCorrelationData": {
      "TestKey1": "TestValue1",
      "testKey2": "testValue2"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

LiveEventConnectionRejected

The following example shows the schema of the **LiveEventConnectionRejected** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaServices/<account-name>",
    "subject": "/LiveEvents/MyLiveEvent1",
    "eventType": "Microsoft.Media.LiveEventConnectionRejected",
    "eventTime": "2018-01-16T01:57:26.005121Z",
    "id": "b303db59-d5c1-47eb-927a-3650875fded1",
    "data": {
      "StreamId": "Mystream1",
      "IngestUrl": "http://abc.ingest.isml",
      "EncoderIp": "118.238.251.xxx",
      "EncoderPort": 52859,
      "ResultCode": "MPE_INGEST_CODEC_NOT_SUPPORTED"
    },
    "dataVersion": "1.0"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
StreamId	string	Identifier of the stream or connection. Encoder or customer is responsible to add this ID in the ingest URL.
IngestUrl	string	Ingest URL provided by the live event.
EncoderIp	string	IP of the encoder.
EncoderPort	string	Port of the encoder from where this stream is coming.
ResultCode	string	The reason the connection was rejected. The result codes are listed in the following table.

The result codes are:

RESULT CODE	DESCRIPTION
MPE_RTMP_APPID_AUTH_FAILURE	Incorrect ingest URL
MPE_INGEST_ENCODER_CONNECTION_DENIED	Encoder IP isn't present in IP allow list configured
MPE_INGEST_RTMP_SETDATAFRAME_NOT_RECEIVED	Encoder didn't send metadata about the stream.
MPE_INGEST_CODEC_NOT_SUPPORTED	Codec specified isn't supported.
MPE_INGEST_DESCRIPTION_INFO_NOT_RECEIVED	Received a fragment before receiving and header for that stream.
MPE_INGEST_MEDIA_QUALITIES_EXCEEDED	Number of qualities specified exceeds allowed max limit.
MPE_INGEST_BITRATE_AGGREGATED_EXCEEDED	Aggregated bitrate exceeds max allowed limit.

RESULT CODE	DESCRIPTION
MPE_RTMP_FLV_TAG_TIMESTAMP_INVALID	The timestamp for video or audio FLVTag is invalid from RTMP encoder.

LiveEventEncoderConnected

The following example shows the schema of the **LiveEventEncoderConnected** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventEncoderConnected",
    "eventTime": "2018-08-07T23:08:09.1710643",
    "id": "<id>",
    "data": {
      "ingestUrl": "http://mle1-amsts03mediaacctgndos-ts031.channel.media.azure-test.net:80/ingest.isml",
      "streamId": "15864-stream0",
      "encoderIp": "131.107.147.xxx",
      "encoderPort": "27485"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
StreamId	string	Identifier of the stream or connection. Encoder or customer is responsible for providing this ID in the ingest URL.
IngestUrl	string	Ingest URL provided by the live event.
EncoderIp	string	IP of the encoder.
EncoderPort	string	Port of the encoder from where this stream is coming.

LiveEventEncoderDisconnected

The following example shows the schema of the **LiveEventEncoderDisconnected** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventEncoderDisconnected",
    "eventTime": "2018-08-07T23:08:09.1710872",
    "id": "<id>",
    "data": {
      "ingestUrl": "http://mle1-amsts03mediaacctgndos-ts031.channel.media.azure-test.net:80/ingest.isml",
      "streamId": "15864-stream0",
      "encoderIp": "131.107.147.xxx",
      "encoderPort": "27485",
      "resultCode": "S_OK"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
StreamId	string	Identifier of the stream or connection. Encoder or customer is responsible to add this ID in the ingest URL.
IngestUrl	string	Ingest URL provided by the live event.
EncoderIp	string	IP of the encoder.
EncoderPort	string	Port of the encoder from where this stream is coming.
ResultCode	string	The reason for the encoder disconnecting. It could be graceful disconnect or from an error. The result codes are listed in the following table.

The error result codes are:

RESULT CODE	DESCRIPTION
MPE_RTMP_SESSION_IDLE_TIMEOUT	RTMP session timed out after being idle for allowed time limit.
MPE_RTMP_FLV_TAG_TIMESTAMP_INVALID	The timestamp for video or audio FLVTag is invalid from RTMP encoder.
MPE_CAPACITY_LIMIT_REACHED	Encoder sending data too fast.
Unknown Error Codes	These error codes can range from memory error to duplicate entries in hash map.

The graceful disconnect result codes are:

RESULT CODE	DESCRIPTION
S_OK	Encoder disconnected successfully.
MPE_CLIENT_TERMINATED_SESSION	Encoder disconnected (RTMP).
MPE_CLIENT_DISCONNECTED	Encoder disconnected (FMP4).
MPI_REST_API_CHANNEL_RESET	Channel reset command is received.
MPI_REST_API_CHANNEL_STOP	Channel stop command received.
MPI_REST_API_CHANNEL_STOP	Channel undergoing maintenance.
MPI_STREAM_HIT_EOF	EOF stream is sent by the encoder.

LiveEventIncomingDataChunkDropped

The following example shows the schema of the **LiveEventIncomingDataChunkDropped** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaServices/<account-name>",
    "subject": "/LiveEvents/MyLiveEvent1",
    "eventType": "Microsoft.Media.LiveEventIncomingDataChunkDropped",
    "eventTime": "2018-01-16T01:57:26.005121Z",
    "id": "03da9c10-fde7-48e1-80d8-49936f2c3e7d",
    "data": {
      "TrackType": "Video",
      "TrackName": "Video",
      "Bitrate": 300000,
      "Timestamp": 36656620000,
      "Timescale": 10000000,
      "ResultCode": "FragmentDrop_OverlapTimestamp"
    },
    "dataVersion": "1.0"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
TrackType	string	Type of the track (Audio / Video).
TrackName	string	Name of the track.
Bitrate	integer	Bitrate of the track.
Timestamp	string	Timestamp of the data chunk dropped.
Timescale	string	Timescale of the timestamp.

PROPERTY	TYPE	DESCRIPTION
ResultCode	string	Reason of the data chunk drop. FragmentDrop_OverlapTimestamp or FragmentDrop_NonIncreasingTimestamp .

LiveEventIncomingStreamReceived

The following example shows the schema of the **LiveEventIncomingStreamReceived** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventIncomingStreamReceived",
    "eventTime": "2018-08-07T23:08:10.5069288Z",
    "id": "7f939a08-320c-47e7-8250-43dcfc04ab4d",
    "data": {
      "ingestUrl": "http://mle1-amsts03mediaacctgndos-ts031.channel.media.azure-test.net:80/ingest.isml/Streams(15864-stream0)15864-stream0",
      "trackType": "video",
      "trackName": "video",
      "bitrate": 2962000,
      "encoderIp": "131.107.147.xxx",
      "encoderPort": "27485",
      "timestamp": "15336831655032322",
      "duration": "20000000",
      "timescale": "10000000"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
TrackType	string	Type of the track (Audio / Video).
TrackName	string	Name of the track (either provided by the encoder or, in case of RTMP, server generates in <i>TrackType_Bitrate</i> format).
Bitrate	integer	Bitrate of the track.
IngestUrl	string	Ingest URL provided by the live event.
EncoderIp	string	IP of the encoder.
EncoderPort	string	Port of the encoder from where this stream is coming.
Timestamp	string	First timestamp of the data chunk received.

PROPERTY	TYPE	DESCRIPTION
Timescale	string	Timescale in which timestamp is represented.

LiveEventIncomingStreamsOutOfSync

The following example shows the schema of the **LiveEventIncomingStreamsOutOfSync** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventIncomingStreamsOutOfSync",
    "eventTime": "2018-08-10T02:26:20.6269183Z",
    "id": "b9d38923-9210-4c2b-958f-0054467d4dd7",
    "data": {
      "minLastTimestamp": "319996",
      "typeOfStreamWithMinLastTimestamp": "Audio",
      "maxLastTimestamp": "366000",
      "typeOfStreamWithMaxLastTimestamp": "Video",
      "timescaleOfMinLastTimestamp": "10000000",
      "timescaleOfMaxLastTimestamp": "10000000"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
MinLastTimestamp	string	Minimum of last timestamps among all the tracks (audio or video).
TypeOfTrackWithMinLastTimestamp	string	Type of the track (audio or video) with minimum last timestamp.
MaxLastTimestamp	string	Maximum of all the timestamps among all the tracks (audio or video).
TypeOfTrackWithMaxLastTimestamp	string	Type of the track (audio or video) with maximum last timestamp.
TimescaleOfMinLastTimestamp	string	Gets the timescale in which "MinLastTimestamp" is represented.
TimescaleOfMaxLastTimestamp	string	Gets the timescale in which "MaxLastTimestamp" is represented.

LiveEventIncomingVideoStreamsOutOfSync

The following example shows the schema of the **LiveEventIncomingVideoStreamsOutOfSync** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaServices/<account-name>",
    "subject": "/LiveEvents/LiveEvent1",
    "eventType": "Microsoft.Media.LiveEventIncomingVideoStreamsOutOfSync",
    "eventTime": "2018-01-16T01:57:26.005121Z",
    "id": "6dd4d862-d442-40a0-b9f3-fc14bcf6d750",
    "data": {
      "FirstTimestamp": "2162058216",
      "FirstDuration": "2000",
      "SecondTimestamp": "2162057216",
      "SecondDuration": "2000",
      "timescale": "10000000"
    },
    "dataVersion": "1.0"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
FirstTimestamp	string	Timestamp received for one of the tracks/quality levels of type video.
FirstDuration	string	Duration of the data chunk with first timestamp.
SecondTimestamp	string	Timestamp received for some other track/quality level of the type video.
SecondDuration	string	Duration of the data chunk with second timestamp.
Timescale	string	Timescale of timestamps and duration.

LiveEventIngestHeartbeat

The following example shows the schema of the **LiveEventIngestHeartbeat** event:


```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventIngestHeartbeat",
    "eventTime": "2018-08-07T23:17:57.4610506",
    "id": "7f450938-491f-41e1-b06f-c6cd3965d786",
    "data": {
      "trackType": "audio",
      "trackName": "audio",
      "bitrate": 160000,
      "incomingBitrate": 155903,
      "lastTimestamp": "15336837535253637",
      "timescale": "10000000",
      "overlapCount": 0,
      "discontinuityCount": 0,
      "nonincreasingCount": 0,
      "unexpectedBitrate": false,
      "state": "Running",
      "healthy": true
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
TrackType	string	Type of the track (Audio / Video).
TrackName	string	Name of the track (either provided by the encoder or, in case of RTMP, server generates in <i>TrackType_Bitrate</i> format).
Bitrate	integer	Bitrate of the track.
IncomingBitrate	integer	Calculated bitrate based on data chunks coming from encoder.
LastTimestamp	string	Latest timestamp received for a track in last 20 seconds.
Timescale	string	Timescale in which timestamps are expressed.
OverlapCount	integer	Number of data chunks had overlapped timestamps in last 20 seconds.
DiscontinuityCount	integer	Number of discontinuities observed in last 20 seconds.
NonIncreasingCount	integer	Number of data chunks with timestamps in the past were received in last 20 seconds.

PROPERTY	TYPE	DESCRIPTION
UnexpectedBitrate	bool	If expected and actual bitrates differ by more than allowed limit in last 20 seconds. It's true if and only if, IncomingBitrate >= 2* bitrate OR IncomingBitrate <= bitrate/2 OR IncomingBitrate = 0.
State	string	State of the live event.
Healthy	bool	Indicates whether ingest is healthy based on the counts and flags. Healthy is true if OverlapCount = 0 && DiscontinuityCount = 0 && NonIncreasingCount = 0 && UnexpectedBitrate = false.

LiveEventTrackDiscontinuityDetected

The following example shows the schema of the **LiveEventTrackDiscontinuityDetected** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventTrackDiscontinuityDetected",
    "eventTime": "2018-08-07T23:18:06.1270405Z",
    "id": "5f4c510d-5be7-4bef-baf0-64b828be9c9b",
    "data": {
      "trackName": "video",
      "previousTimestamp": "15336837615032322",
      "trackType": "video",
      "bitrate": 2962000,
      "newTimestamp": "15336837619774273",
      "discontinuityGap": "575284",
      "timescale": "10000000"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
TrackType	string	Type of the track (Audio / Video).
TrackName	string	Name of the track (either provided by the encoder or, in case of RTMP, server generates in <i>TrackType_Bitrate</i> format).
Bitrate	integer	Bitrate of the track.
PreviousTimestamp	string	Timestamp of the previous fragment.
NewTimestamp	string	Timestamp of the current fragment.

PROPERTY	TYPE	DESCRIPTION
DiscontinuityGap	string	Gap between above two timestamps.
Timescale	string	Timescale in which both timestamp and discontinuity gap are represented.

Common event properties

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
topic	string	The EventGrid topic. This property has the resource ID for the Media Services account.
subject	string	The resource path for the Media Services channel under the Media Services account. Concatenating the topic and subject give you the resource ID for the job.
eventType	string	One of the registered event types for this event source. For example, "Microsoft.Media.JobStateChange".
eventTime	string	The time the event is generated based on the provider's UTC time.
id	string	Unique identifier for the event.
data	object	Media Services event data.
dataVersion	string	The schema version of the data object. The publisher defines the schema version.
metadataVersion	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

Next steps

[Register for job state change events](#)

See also

- [EventGrid .NET SDK that includes Media Service events](#)
- [Definitions of Media Services events](#)

Media Services PlayReady license template overview

10/17/2018 • 5 minutes to read • [Edit Online](#)

Azure Media Services enables you to encrypt your content with **Microsoft PlayReady**. Media Services also provides a service for delivering PlayReady licenses. You can use Media Services APIs to configure PlayReady licenses. When a player tries to play your PlayReady-protected content, a request is sent to the license delivery service to obtain a license. If the license service approves the request, it issues the license that is sent to the client and is used to decrypt and play the specified content.

PlayReady licenses contain the rights and restrictions that you want the PlayReady digital rights management (DRM) runtime to enforce when a user tries to play back protected content. Here are some examples of PlayReady license restrictions that you can specify:

- The date and time from which the license is valid.
- The DateTime value when the license expires.
- For the license to be saved in persistent storage on the client. Persistent licenses are typically used to allow offline playback of the content.
- The minimum security level that a player must have to play your content.
- The output protection level for the output controls for audio\video content.
- For more information, see the "Output Controls" section (3.5) in the [PlayReady Compliance Rules](#) document.

NOTE

Currently, you can only configure the PlayRight of the PlayReady license. This right is required. The PlayRight gives the client the ability to play back the content. You also can use the PlayRight to configure restrictions specific to playback.

This topic describes how to configure PlayReady licenses with Media Services.

Basic streaming license example

The following example shows the simplest (and most common) template that configures a basic streaming license. With this license, your clients can play back your PlayReady-protected content.

The XML conforms to the PlayReady license template XML schema defined in the [PlayReady license template XML schema](#) section.

```
<?xml version="1.0" encoding="utf-8"?>
<PlayReadyLicenseResponseTemplate xmlns:i="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://schemas.microsoft.com/Azure/MediaServices/KeyDelivery/PlayReadyTemplate/v1">
  <LicenseTemplates>
    <PlayReadyLicenseTemplate>
      <ContentKey i:type="ContentEncryptionKeyFromHeader" />
      <PlayRight />
    </PlayReadyLicenseTemplate>
  </LicenseTemplates>
</PlayReadyLicenseResponseTemplate>
```

Use Media Services APIs to configure license templates

Media Services provides types that you can use to configure a PlayReady license template.

The snippet that follows uses Media Services .NET classes to configure the PlayReady license template. The classes are defined in the [Microsoft.Azure.Management.Media.Models](#) namespace. The snippet configures the PlayRight of the PlayReady license. PlayRight grants the user the ability to play back the content subject to any restrictions configured in the license and on the PlayRight itself (for playback-specific policy). Much of the policy on a PlayRight concerns output restriction that control the types of outputs that the content can be played over. It also includes any restrictions that must be put in place when a given output is used. For example, if DigitalVideoOnlyContentRestriction is enabled, the DRM runtime only allows the video to be displayed over digital outputs. (Analog video outputs aren't allowed to pass the content.)

IMPORTANT

PlayReady license has restrictions that are powerful. If the output protections are too restrictive, the content might be unplayable on some clients. For more information, see the [PlayReady Compliance Rules](#).

Configure PlayReady license template with .NET

```
ContentKeyPolicyPlayReadyLicense objContentKeyPolicyPlayReadyLicense;
objContentKeyPolicyPlayReadyLicense = new ContentKeyPolicyPlayReadyLicense
{
    AllowTestDevices = true,
    BeginDate = new DateTime(2016, 1, 1),
    ContentKeyLocation = new ContentKeyPolicyPlayReadyContentEncryptionKeyFromHeader(),
    ContentType = ContentKeyPolicyPlayReadyContentType.UltraVioletStreaming,
    LicenseType = drmSettings.EnableOfflineMode ? ContentKeyPolicyPlayReadyLicenseType.Persistent :
ContentKeyPolicyPlayReadyLicenseType.NonPersistent,
    PlayRight = new ContentKeyPolicyPlayReadyPlayRight
    {
        ImageConstraintForAnalogComponentVideoRestriction = true,
        ExplicitAnalogTelevisionOutputRestriction = new
ContentKeyPolicyPlayReadyExplicitAnalogTelevisionRestriction(true, 2),
        AllowPassingVideoContentToUnknownOutput = ContentKeyPolicyPlayReadyUnknownOutputPassingOption.Allowed,
        FirstPlayExpiration = TimeSpan.FromSeconds(20.0),
    }
};
```

PlayReady license template XML schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:tns="http://schemas.microsoft.com/Azure/MediaServices/KeyDelivery/PlayReadyTemplate/v1"
xmlns:ser="http://schemas.microsoft.com/2003/10/Serialization/" elementFormDefault="qualified"
targetNamespace="http://schemas.microsoft.com/Azure/MediaServices/KeyDelivery/PlayReadyTemplate/v1"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="http://schemas.microsoft.com/2003/10/Serialization/" />
  <xs:complexType name="AgcAndColorStripeRestriction">
    <xs:sequence>
      <xs:element minOccurs="0" name="ConfigurationData" type="xs:unsignedByte" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="AgcAndColorStripeRestriction" nillable="true" type="tns:AgcAndColorStripeRestriction" />
  <xs:simpleType name="ContentType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Unspecified" />
      <xs:enumeration value="UltravioletDownload" />
      <xs:enumeration value="UltravioletStreaming" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="ContentType" nillable="true" type="tns:ContentType" />
  <xs:complexType name="ExplicitAnalogTelevisionRestriction">
    <xs:sequence>
      <xs:element minOccurs="0" name="BestEffort" type="xs:boolean" />
      <xs:element minOccurs="0" name="ConfigurationData" type="xs:unsignedByte" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="ExplicitAnalogTelevisionRestriction" nillable="true" type="tns:ExplicitAnalogTelevisionRestriction" />
```

```

    </xs:sequence>
  </xs:complexType>
  <xs:element name="ExplicitAnalogTelevisionRestriction" nillable="true"
type="tns:ExplicitAnalogTelevisionRestriction" />
  <xs:complexType name="PlayReadyContentKey">
    <xs:sequence />
  </xs:complexType>
  <xs:element name="PlayReadyContentKey" nillable="true" type="tns:PlayReadyContentKey" />
  <xs:complexType name="ContentEncryptionKeyFromHeader">
    <xs:complexContent mixed="false">
      <xs:extension base="tns:PlayReadyContentKey">
        <xs:sequence />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="ContentEncryptionKeyFromHeader" nillable="true" type="tns:ContentEncryptionKeyFromHeader"
/>
  <xs:complexType name="ContentEncryptionKeyFromKeyIdentifier">
    <xs:complexContent mixed="false">
      <xs:extension base="tns:PlayReadyContentKey">
        <xs:sequence>
          <xs:element minOccurs="0" name="KeyIdentifier" type="ser:guid" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="ContentEncryptionKeyFromKeyIdentifier" nillable="true"
type="tns:ContentEncryptionKeyFromKeyIdentifier" />
  <xs:complexType name="PlayReadyLicenseResponseTemplate">
    <xs:sequence>
      <xs:element name="LicenseTemplates" nillable="true" type="tns:ArrayOfPlayReadyLicenseTemplate" />
      <xs:element minOccurs="0" name="ResponseCustomData" nillable="true" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="PlayReadyLicenseResponseTemplate" nillable="true"
type="tns:PlayReadyLicenseResponseTemplate" />
  <xs:complexType name="ArrayOfPlayReadyLicenseTemplate">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="PlayReadyLicenseTemplate" nillable="true"
type="tns:PlayReadyLicenseTemplate" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="ArrayOfPlayReadyLicenseTemplate" nillable="true"
type="tns:ArrayOfPlayReadyLicenseTemplate" />
  <xs:complexType name="PlayReadyLicenseTemplate">
    <xs:sequence>
      <xs:element minOccurs="0" name="AllowTestDevices" type="xs:boolean" />
      <xs:element minOccurs="0" name="BeginDate" nillable="true" type="xs:dateTime">
        <xs:annotation>
          <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="ContentKey" nillable="true" type="tns:PlayReadyContentKey" />
  <xs:element minOccurs="0" name="ContentType" type="tns:ContentType">
    <xs:annotation>
      <xs:appinfo>
        <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>

```

```

        </xs:annotation>
    </xs:element>
    <xs:element minOccurs="0" name="ExpirationDate" nillable="true" type="xs:dateTime">
        <xs:annotation>
            <xs:appinfo>
                <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
            </xs:appinfo>
        </xs:annotation>
    </xs:element>
    <xs:element minOccurs="0" name="GracePeriod" nillable="true" type="ser:duration">
        <xs:annotation>
            <xs:appinfo>
                <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
            </xs:appinfo>
        </xs:annotation>
    </xs:element>
    <xs:element minOccurs="0" name="LicenseType" type="tns:PlayReadyLicenseType" />
    <xs:element minOccurs="0" name="PlayRight" nillable="true" type="tns:PlayReadyPlayRight" />
</xs:sequence>
</xs:complexType>
<xs:element name="PlayReadyLicenseTemplate" nillable="true" type="tns:PlayReadyLicenseTemplate" />
<xs:simpleType name="PlayReadyLicenseType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Nonpersistent" />
        <xs:enumeration value="Persistent" />
    </xs:restriction>
</xs:simpleType>
<xs:element name="PlayReadyLicenseType" nillable="true" type="tns:PlayReadyLicenseType" />
<xs:complexType name="PlayReadyPlayRight">
    <xs:sequence>
        <xs:element minOccurs="0" name="AgcAndColorStripeRestriction" nillable="true"
type="tns:AgcAndColorStripeRestriction">
            <xs:annotation>
                <xs:appinfo>
                    <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
                </xs:appinfo>
            </xs:annotation>
        </xs:element>
        <xs:element minOccurs="0" name="AllowPassingVideoContentToUnknownOutput"
type="tns:UnknownOutputPassingOption">
            <xs:annotation>
                <xs:appinfo>
                    <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
                </xs:appinfo>
            </xs:annotation>
        </xs:element>
        <xs:element minOccurs="0" name="AnalogVideoOpl" nillable="true" type="xs:int">
            <xs:annotation>
                <xs:appinfo>
                    <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
                </xs:appinfo>
            </xs:annotation>
        </xs:element>
        <xs:element minOccurs="0" name="CompressedDigitalAudioOpl" nillable="true" type="xs:int">
            <xs:annotation>
                <xs:appinfo>
                    <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
                </xs:appinfo>
            </xs:annotation>
        </xs:element>
        <xs:element minOccurs="0" name="CompressedDigitalVideoOpl" nillable="true" type="xs:int">
            <xs:annotation>
                <xs:appinfo>
                    <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
                </xs:appinfo>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

```

```

        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="DigitalVideoOnlyContentRestriction" type="xs:boolean">
    <xs:annotation>
        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="ExplicitAnalogTelevisionOutputRestriction" nillable="true"
type="tns:ExplicitAnalogTelevisionRestriction">
    <xs:annotation>
        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="FirstPlayExpiration" nillable="true" type="ser:duration">
    <xs:annotation>
        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="ImageConstraintForAnalogComponentVideoRestriction" type="xs:boolean">
    <xs:annotation>
        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="ImageConstraintForAnalogComputerMonitorRestriction" type="xs:boolean">
    <xs:annotation>
        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="ScmsRestriction" nillable="true" type="tns:ScmsRestriction">
    <xs:annotation>
        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="UncompressedDigitalAudioOpl" nillable="true" type="xs:int">
    <xs:annotation>
        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>
<xs:element minOccurs="0" name="UncompressedDigitalVideoOpl" nillable="true" type="xs:int">
    <xs:annotation>
        <xs:appinfo>
            <DefaultValue EmitDefaultValue="false" xmlns="http://schemas.microsoft.com/2003/10/Serialization/"
/>
        </xs:appinfo>
    </xs:annotation>
</xs:element>

```



```

        </xs:annotation>
    </xs:element>
</xs:sequence>
</xs:complexType>
<xs:element name="PlayReadyPlayRight" nillable="true" type="tns:PlayReadyPlayRight" />
<xs:simpleType name="UnknownOutputPassingOption">
    <xs:restriction base="xs:string">
        <xs:enumeration value="NotAllowed" />
        <xs:enumeration value="Allowed" />
        <xs:enumeration value="AllowedWithVideoConstriction" />
    </xs:restriction>
</xs:simpleType>
<xs:element name="UnknownOutputPassingOption" nillable="true" type="tns:UnknownOutputPassingOption" />
<xs:complexType name="ScmsRestriction">
    <xs:sequence>
        <xs:element minOccurs="0" name="ConfigurationData" type="xs:unsignedByte" />
    </xs:sequence>
</xs:complexType>
<xs:element name="ScmsRestriction" nillable="true" type="tns:ScmsRestriction" />
</xs:schema>

```

Next steps

Check out how to [protect with DRM](#)

Widevine license template overview

10/17/2018 • 7 minutes to read • [Edit Online](#)

Azure Media Services enables you to encrypt your content with **Google Widevine**. Media Services also provides a service for delivering Widevine licenses. You can use Azure Media Services APIs to configure Widevine licenses. When a player tries to play your Widevine-protected content, a request is sent to the license delivery service to obtain the license. If the license service approves the request, the service issues the license. It's sent to the client and is used to decrypt and play the specified content.

A Widevine license request is formatted as a JSON message.

NOTE

You can create an empty message with no values, just "{}." Then a license template is created with defaults. The default works for most cases. Microsoft-based license-delivery scenarios should always use the defaults. If you need to set the "provider" and "content_id" values, a provider must match Widevine credentials.

```
{
  "payload": "<license challenge>",
  "content_id": "<content id>"
  "provider": "<provider>"
  "allowed_track_types": "<types>",
  "content_key_specs": [
    {
      "track_type": "<track type 1>"
    },
    {
      "track_type": "<track type 2>"
    },
    ...
  ],
  "policy_overrides": {
    "can_play": <can play>,
    "can_persist": <can persist>,
    "can_renew": <can renew>,
    "rental_duration_seconds": <rental duration>,
    "playback_duration_seconds": <playback duration>,
    "license_duration_seconds": <license duration>,
    "renewal_recovery_duration_seconds": <renewal recovery duration>,
    "renewal_server_url": "<renewal server url>",
    "renewal_delay_seconds": <renewal delay>,
    "renewal_retry_interval_seconds": <renewal retry interval>,
    "renew_with_usage": <renew with usage>
  }
}
```

JSON message

NAME	VALUE	DESCRIPTION
payload	Base64-encoded string	The license request sent by a client.

NAME	VALUE	DESCRIPTION
content_id	Base64-encoded string	Identifier used to derive the key ID and content key for each content_key_specs.track_type.
provider	string	Used to look up content keys and policies. If Microsoft key delivery is used for Widevine license delivery, this parameter is ignored.
policy_name	string	Name of a previously registered policy. Optional.
allowed_track_types	enum	SD_ONLY or SD_HD. Controls which content keys are included in a license.
content_key_specs	Array of JSON structures, see the section "Content key specs."	A finer-grained control on which content keys to return. For more information, see the section "Content key specs." Only one of the allowed_track_types and content_key_specs values can be specified.
use_policy_overrides_exclusively	Boolean, true or false	Use policy attributes specified by policy_overrides, and omit all previously stored policy.
policy_overrides	JSON structure, see the section "Policy overrides."	Policy settings for this license. In the event this asset has a predefined policy, these specified values are used.
session_init	JSON structure, see the section "Session initialization."	Optional data is passed to the license.
parse_only	Boolean, true or false	The license request is parsed, but no license is issued. However, values from the license request are returned in the response.

Content key specs

If a pre-existing policy exists, there is no need to specify any of the values in the content key spec. The pre-existing policy associated with this content is used to determine the output protection, such as High-bandwidth Digital Content Protection (HDCP) and the Copy General Management System (CGMS). If a pre-existing policy isn't registered with the Widevine license server, the content provider can inject the values into the license request.

Each content_key_specs value must be specified for all tracks, regardless of the use_policy_overrides_exclusively option.

NAME	VALUE	DESCRIPTION
------	-------	-------------

NAME	VALUE	DESCRIPTION
content_key_specs.track_type	string	A track type name. If content_key_specs is specified in the license request, make sure to specify all track types explicitly. Failure to do so results in failure to play back past 10 seconds.
content_key_specs.security_level	uint32	Defines client robustness requirements for playback. <ul style="list-style-type: none"> - Software-based white-box cryptography is required. - Software cryptography and an obfuscated decoder are required. - The key material and cryptography operations must be performed within a hardware-backed trusted execution environment. - The cryptography and decoding of content must be performed within a hardware-backed trusted execution environment. - The cryptography, decoding, and all handling of the media (compressed and uncompressed) must be handled within a hardware-backed trusted execution environment.
content_key_specs.required_output_protection.hdc	string, one of HDCP_NONE, HDCP_V1, HDCP_V2	Indicates whether HDCP is required.
content_key_specs.key	Base64-encoded string	Content key to use for this track. If specified, the track_type or key_id is required. The content provider can use this option to inject the content key for this track instead of letting the Widevine license server generate or look up a key.
content_key_specs.key_id	Base64-encoded string binary, 16 bytes	Unique identifier for the key.

Policy overrides

NAME	VALUE	DESCRIPTION
policy_overrides.can_play	Boolean, true or false	Indicates that playback of the content is allowed. Default is false.
policy_overrides.can_persist	Boolean, true or false	Indicates that the license might be persisted to nonvolatile storage for offline use. Default is false.
policy_overrides.can_renew	Boolean, true or false	Indicates that renewal of this license is allowed. If true, the duration of the license can be extended by heartbeat. Default is false.

NAME	VALUE	DESCRIPTION
policy_overrides.license_duration_seconds	int64	Indicates the time window for this specific license. A value of 0 indicates that there is no limit to the duration. Default is 0.
policy_overrides.rental_duration_seconds	int64	Indicates the time window while playback is permitted. A value of 0 indicates that there is no limit to the duration. Default is 0.
policy_overrides.playback_duration_seconds	int64	The viewing window of time after playback starts within the license duration. A value of 0 indicates that there is no limit to the duration. Default is 0.
policy_overrides.renewal_server_url	string	All heartbeat (renewal) requests for this license is directed to the specified URL. This field is used only if can_renew is true.
policy_overrides.renewal_delay_seconds	int64	How many seconds after license_start_time before renewal is first attempted. This field is used only if can_renew is true. Default is 0.
policy_overrides.renewal_retry_interval_seconds	int64	Specifies the delay in seconds between subsequent license renewal requests, in case of failure. This field is used only if can_renew is true.
policy_overrides.renewal_recovery_duration_seconds	int64	The window of time in which playback can continue while renewal is attempted, yet unsuccessful due to back-end problems with the license server. A value of 0 indicates that there is no limit to the duration. This field is used only if can_renew is true.
policy_overrides.renew_with_usage	Boolean, true or false	Indicates that the license is sent for renewal when usage starts. This field is used only if can_renew is true.

Session initialization

NAME	VALUE	DESCRIPTION
provider_session_token	Base64-encoded string	This session token is passed back in the license and exists in subsequent renewals. The session token doesn't persist beyond sessions.

NAME	VALUE	DESCRIPTION
provider_client_token	Base64-encoded string	Client token to send back in the license response. If the license request contains a client token, this value is ignored. The client token persists beyond license sessions.
override_provider_client_token	Boolean, true or false	If false and the license request contains a client token, use the token from the request even if a client token was specified in this structure. If true, always use the token specified in this structure.

Configure your Widevine license with .NET

Media Services provides a class that lets you configure a Widevine license. To construct the license, pass JSON to [WidevineTemplate](#).

To configure the template, you can:

Directly construct a JSON string

This method may be error-prone. It is recommended to use other method, described in [Define needed classes and serialize to JSON](#).

```

` ` ` csharp
ContentKeyPolicyWidevineConfiguration objContentKeyPolicyWidevineConfiguration = new
ContentKeyPolicyWidevineConfiguration
{
    WidevineTemplate = @"{"allowed_track_types":"","SD_HD","content_key_specs":
[{"track_type":"","SD","security_level":1,"required_output_protection":
{"hdcp":"","HDCP_V2"}}, {"policy_overrides":{"can_play":true,"can_persist":true,"can_renew":false}}
];
` ` `

```

Define needed classes and serialize to JSON

Define classes

The following example shows an example of definitions of classes that map to Widevine JSON schema. You can instantiate the classes before serializing them to JSON string.

```

```csharp
public class PolicyOverrides
{
 public bool CanPlay { get; set; }
 public bool CanPersist { get; set; }
 public bool CanRenew { get; set; }
 public int RentalDurationSeconds { get; set; } //Indicates the time window while playback is permitted.
 A value of 0 indicates that there is no limit to the duration. Default is 0.
 public int PlaybackDurationSeconds { get; set; } //The viewing window of time after playback starts
 within the license duration. A value of 0 indicates that there is no limit to the duration. Default is 0.
 public int LicenseDurationSeconds { get; set; } //Indicates the time window for this specific license. A
 value of 0 indicates that there is no limit to the duration. Default is 0.
}

public class ContentKeySpec
{
 public string TrackType { get; set; }
 public int SecurityLevel { get; set; }
 public OutputProtection RequiredOutputProtection { get; set; }
}

public class OutputProtection
{
 public string HDCP { get; set; }
}

public class WidevineTemplate
{
 public string AllowedTrackTypes { get; set; }
 public ContentKeySpec[] ContentKeySpecs { get; set; }
 public PolicyOverrides PolicyOverrides { get; set; }
}
```

```

Configure the license

Use classes defined in the previous section to create JSON that is used to configure [WidevineTemplate](#):

```

private static ContentKeyPolicyWidevineConfiguration ConfigureWidevineLicenseTempate()
{
    WidevineTemplate template = new WidevineTemplate()
    {
        AllowedTrackTypes = "SD_HD",
        ContentKeySpecs = new ContentKeySpec[]
        {
            new ContentKeySpec()
            {
                TrackType = "SD",
                SecurityLevel = 1,
                RequiredOutputProtection = new OutputProtection()
                {
                    HDCP = "HDCP_V2"
                }
            }
        },
        PolicyOverrides = new PolicyOverrides()
        {
            CanPlay = true,
            CanPersist = true,
            CanRenew = false,
            RentalDurationSeconds = 2592000,
            PlaybackDurationSeconds = 10800,
            LicenseDurationSeconds = 604800,
        }
    };

    ContentKeyPolicyWidevineConfiguration objContentKeyPolicyWidevineConfiguration = new
ContentKeyPolicyWidevineConfiguration
    {
        WidevineTemplate = Newtonsoft.Json.JsonConvert.SerializeObject(template)
    };
    return objContentKeyPolicyWidevineConfiguration;
}

```

Next steps

Check out how to [protect with DRM](#)

Apple FairPlay license requirements and configuration

12/10/2018 • 4 minutes to read • [Edit Online](#)

Azure Media Services enables you to encrypt your HLS content with **Apple FairPlay** (AES-128 CBC). Media Services also provides a service for delivering FairPlay licenses. When a player tries to play your FairPlay-protected content, a request is sent to the license delivery service to obtain a license. If the license service approves the request, it issues the license that is sent to the client and is used to decrypt and play the specified content.

Media Services also provides APIs that you can use to configure your FairPlay licenses. This topic discusses FairPlay license requirements and demonstrates how you can configure a **FairPlay** license using Media Services APIs.

Requirements

The following are required when using Media Services to encrypt your HLS content with **Apple FairPlay** and use Media Services to deliver FairPlay licenses:

- Sign up with [Apple Development Program](#).
- Apple requires the content owner to obtain the [deployment package](#). State that you already implemented Key Security Module (KSM) with Media Services, and that you are requesting the final FPS package. There are instructions in the final FPS package to generate certification and obtain the Application Secret Key (ASK). You use ASK to configure FairPlay.
- The following things must be set on Media Services key/license delivery side:
 - **App Cert (AC)**: This is a .pfx file that contains the private key. You create this file and encrypt it with a password. The .pfx file should be in Base64 format.

The following steps describe how to generate a .pfx certificate file for FairPlay:

1. Install OpenSSL from <https://slproweb.com/products/Win32OpenSSL.html>.

Go to the folder where the FairPlay certificate and other files delivered by Apple are.

2. Run the following command from the command line. This converts the .cer file to a .pem file.

```
"C:\OpenSSL-Win32\bin\openssl.exe" x509 -inform der -in FairPlay.cer -out FairPlay-out.pem
```

3. Run the following command from the command line. This converts the .pem file to a .pfx file with the private key. The password for the .pfx file is then asked by OpenSSL.

```
"C:\OpenSSL-Win32\bin\openssl.exe" pkcs12 -export -out FairPlay-out.pfx -inkey privatekey.pem -in FairPlay-out.pem -passin file:privatekey-pem-pass.txt
```

- **App Cert password**: The password for creating the .pfx file.
 - **ASK**: This key is received when you generate the certification by using the Apple Developer portal. Each development team receives a unique ASK. Save a copy of the ASK, and store it in a safe place. You need to configure ASK as FairPlayAsk with Media Services.
- The following things must be set by the FPS client side:

- **App Cert (AC)**: This is a .cer/.der file that contains the public key, which the operating system uses to encrypt some payload. Media Services needs to know about it because it is required by the player. The key delivery service decrypts it using the corresponding private key.
- To play back a FairPlay encrypted stream, get a real ASK first, and then generate a real certificate. That process creates all three parts:
 - .der file
 - .pfx file
 - password for the .pfx

FairPlay and player apps

When your content is encrypted with **Apple FairPlay**, the individual video and audio samples are encrypted by using the **AES-128 CBC** mode. **FairPlay Streaming** (FPS) is integrated into the device operating systems, with native support on iOS and Apple TV. Safari on OS X enables FPS by using the Encrypted Media Extensions (EME) interface support.

Azure Media Player also supports FairPlay playback. For more information, see [Azure Media Player documentation](#).

You can develop your own player apps by using the iOS SDK. To be able to play FairPlay content, you have to implement the license exchange protocol. This protocol is not specified by Apple. It is up to each app how to send key delivery requests. The Media Services FairPlay key delivery service expects the SPC to come as a www-form-urlencoded post message, in the following form:

```
spc=<Base64 encoded SPC>
```

FairPlay configuration .NET example

You can use Media Services API to configure FairPlay licenses. When the player tries to play your FairPlay-protected content, a request is sent to the license delivery service to obtain the license. If the license service approves the request, the service issues the license. It's sent to the client and is used to decrypt and play the specified content.

NOTE

Usually, you would want to configure FairPlay policy options only once, because you will only have one set of a certification and an ASK.

The following example uses [Media Services .NET SDK](#) to configure the license.

```

private static ContentKeyPolicyFairPlayConfiguration ConfigureFairPlayPolicyOptions()
{
    string askHex = "";
    string FairPlayPfxPassword = "";

    var appCert = new X509Certificate2("FairPlayPfxPath", FairPlayPfxPassword,
X509KeyStorageFlags.Exportable);

    byte[] askBytes = Enumerable
        .Range(0, askHex.Length)
        .Where(x => x % 2 == 0)
        .Select(x => Convert.ToByte(askHex.Substring(x, 2), 16))
        .ToArray();

    ContentKeyPolicyFairPlayConfiguration fairPlayConfiguration =
    new ContentKeyPolicyFairPlayConfiguration
    {
        Ask = askBytes,
        FairPlayPfx =
            Convert.ToBase64String(appCert.Export(X509ContentType.Pfx, FairPlayPfxPassword)),
        FairPlayPfxPassword = FairPlayPfxPassword,
        RentalAndLeaseKeyType =
            ContentKeyPolicyFairPlayRentalAndLeaseKeyType
                .PersistentUnlimited,
        RentalDuration = 2249
    };

    return fairPlayConfiguration;
}

```

Next steps

Check out how to [protect with DRM](#)

Azure Media Services v3 release notes

11/26/2018 • 4 minutes to read • [Edit Online](#)

To stay up-to-date with the most recent developments, this article provides you with information about:

- The latest releases
- Known issues
- Bug fixes
- Deprecated functionality
- Plans for changes

Known issues

NOTE

Currently, you cannot use the Azure portal to manage v3 resources. Use the [REST API](#), CLI, or one of the supported SDKs.

For more information, see [Migration guidance for moving from Media Services v2 to v3](#).

November 2018

The CLI 2.0 module is now available for [Azure Media Services v3 GA](#) – v 2.0.50.

New commands

- [az ams account](#)
- [az ams account-filter](#)
- [az ams asset](#)
- [az ams asset-filter](#)
- [az ams content-key-policy](#)
- [az ams job](#)
- [az ams live-event](#)
- [az ams live-output](#)
- [az ams streaming-endpoint](#)
- [az ams streaming-locator](#)
- [az ams account mru](#) - enables you to manage Media Reserved Units

New features and breaking changes

Asset commands

- `--storage-account` and `--container` arguments added.
- Default values for expiry time (Now+23h) and permissions (Read) in `az ams asset get-sas-url` command added.

Job commands

- `--correlation-data` and `--label` arguments added
- `--output-asset-names` renamed to `--output-assets`. Now it accepts a space-separated list of assets in 'assetName=label' format. An asset without label can be sent like this: 'assetName='.

Streaming Locator commands

- `az ams streaming locator` base command replaced with `az ams streaming-locator`.
- `--streaming-locator-id` and `--alternative-media-id` support arguments added.
- `--content-keys` argument argument updated.
- `--content-policy-name` renamed to `--content-key-policy-name`.

Streaming Policy commands

- `az ams streaming policy` base command replaced with `az ams streaming-policy`.
- Encryption parameters support in `az ams streaming-policy create` added.

Transform commands

- `--preset-names` argument replaced with `--preset`. Now you can only set 1 output/preset at a time (to add more you have to run `az ams transform output add`). Also, you can set custom StandardEncoderPreset by passing the path to your custom JSON.
- `az ams transform output remove` can be performed by passing the output index to remove.
- `--relative-priority`, `--on-error`, `--audio-language` and `--insights-to-extract` arguments added in `az ams transform create` and `az ams transform output add` commands.

October 2018 - GA

This section describes Azure Media Services (AMS) October updates.

REST v3 GA release

The [REST v3 GA release](#) includes more APIs for Live, Account/Asset level manifest filters, and DRM support.

Azure Resource Management

Support for Azure Resource Management enables unified management and operations API (now everything in one place).

Starting with this release, you can use Resource Manager templates to create Live Events.

Improvement of Asset operations

The following improvements were introduced:

- Ingest from HTTP(s) URLs or Azure Blob Storage SAS URLs.
- Specify you own container names for Assets.
- Easier output support to create custom workflows with Azure Functions.

New Transform object

The new **Transform** object simplifies the Encoding model. The new object makes it easy to create and share encoding Resource Manager templates and presets.

Azure Active Directory authentication and RBAC

Azure AD Authentication and Role-Based Access Control (RBAC) enable secure Transforms, LiveEvents, Content Key Policies, or Assets by Role or Users in Azure AD.

Client SDKs

Languages supported in Media Services v3: .NET Core, Java, Node.js, Ruby, Typescript, Python, Go.

Live encoding updates

The following live encoding updates are introduced:

- New low latency mode for live (10 seconds end-to-end).
- Improved RTMP support (increased stability and more source encoder support).
- RTMPS secure ingest.

When you create a LiveEvent, you now get 4 ingest URLs. The 4 ingest URLs are almost identical, have the same streaming token (AppId), only the port number part is different. Two of the URLs are primary and

backup for RTMPs.

- 24-hour transcoding support.
- Improved ad-signaling support in RTMP via SCTE35.

Improved Event Grid support

You can see the following Event Grid support improvements:

- Azure EventGrid integration for easier development with Logic Apps and Azure Functions.
- Subscribe for events on Encoding, Live Channels, and more.

CMAF support

CMAF and 'cbcs' encryption support for Apple HLS (iOS 11+) and MPEG-DASH players that support CMAF.

Video Indexer

Video Indexer GA release was announced in August. For new information about currently supported features, see [What is Video Indexer](#).

Plans for changes

Azure CLI 2.0

The Azure CLI 2.0 module that includes operations on all features (including Live, Content Key Policies, Account/Asset Filters, Streaming Policies) is coming soon.

Known issues

Only customers that used the preview API for Asset or AccountFilters are impacted by the following issue.

If you created Assets or Account Filters between 09/28 and 10/12 with Media Services v3 CLI or APIs, you need to remove all Asset and AccountFilters and re-create them due to a version conflict.

May 2018 - Preview

.Net SDK

The following features are present in the .Net SDK:

- **Transforms** and **Jobs** to encode or analyze media content. For examples, see [Stream files](#) and [Analyze](#).
- **StreamingLocators** for publishing and streaming content to end-user devices
- **StreamingPolicies** and **ContentKeyPolicies** to configure key delivery and content protection (DRM) when delivering content.
- **LiveEvents** and **LiveOutputs** to configure the ingest and archiving of live streaming content.
- **Assets** to store and publish media content in Azure Storage.
- **StreamingEndpoints** to configure and scale dynamic packaging, encryption, and streaming for both live and on-demand media content.

Known issues

- When submitting a job, you can specify to ingest your source video using HTTPS URLs, SAS URLs, or paths to files located in Azure Blob storage. Currently, AMS v3 does not support chunked transfer encoding over HTTPS URLs.

Next steps

[Overview](#)

Quotas and limitations in Azure Media Services v3

10/17/2018 • 2 minutes to read • [Edit Online](#)

This article describes quotas and limitations in Azure Media Services v3.

| RESOURCE | DEFAULT LIMIT |
|---|---|
| Assets per Azure Media Services account | 1,000,000 |
| Dynamic Manifest Filters | 100 |
| JobInputs per Job | 50 (fixed) |
| JobOutputs per Job/TransformOutputs in a Transform | 20 (fixed) |
| Files per JobInput | 10 (fixed) |
| File size | In some scenarios, there is a limit on the maximum file size supported for processing in Media Services. ⁽¹⁾ |
| Jobs per Media Services account | 500,000 ⁽²⁾ (fixed) |
| Listing Transforms | Paginate the response, with 1000 Transforms per page |
| Listing Jobs | Paginate the response, with 500 Jobs per page |
| LiveEvents per Media Services account | 5 |
| Media Services accounts in a single subscription | 25 (fixed) |
| LiveOutputs in running state per LiveEvent | 3 |
| Storage accounts | 100 ⁽⁴⁾ (fixed) |
| Streaming Endpoints in running state per Media Services account | 2 |
| StreamingPolicies | 100 ⁽³⁾ |
| Transforms per Media Services account | 100 (fixed) |
| Unique StreamingLocators associated with an Asset at one time | 100 ⁽⁵⁾ (fixed) |

¹ The maximum size supported for a single blob is currently up to 5 TB in Azure Blob Storage. However, additional limits apply in Azure Media Services based on the VM sizes that are used by the service. If your source file is larger than 260-GB, your Job will likely fail. If you have 4K content that is larger than 260-GB limit, contact us at amshelp@microsoft.com for potential mitigations to support your scenario.

² This number includes queued, finished, active, and canceled Jobs. It does not include deleted Jobs.

Any Job record in your account older than 90 days will be automatically deleted, even if the total number of records is below the maximum quota.

³ When using a custom [StreamingPolicy](#), you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed. You should not be creating a new StreamingPolicy for each StreamingLocator.

⁴ The storage accounts must be from the same Azure subscription.

⁵ StreamingLocators are not designed for managing per-user access control. To give different access rights to individual users, use Digital Rights Management (DRM) solutions.

Support ticket

For resources that are not fixed, you may ask for the quotas to be raised, by opening a [support ticket](#). Include detailed information in the request on the desired quota changes, use-case scenarios, and regions required. Do **not** create additional Azure Media Services accounts in an attempt to obtain higher limits.

Next steps

[Overview](#)

Azure Media Services v3 frequently asked questions

11/5/2018 • 2 minutes to read • [Edit Online](#)

This article gives answers to Azure Media Services (AMS) v3 frequently asked questions.

Can I use the Azure portal to manage v3 resources?

Not yet. You can use one of the supported SDKs. See tutorials and samples in this doc set.

Is there an API for configuring Media Reserved Units?

Currently, you have to use AMS v2 APIs to configure media Reserved Units (as described in [Scaling media processing](#)).

When using **VideoAnalyzerPreset** and/or **AudioAnalyzerPreset**, set your Media Services account to 10 S3 Media Reserved Units.

Does V3 Asset have no AssetFile concept?

The AssetFiles were removed from the AMS API in order to separate Media Services from Storage SDK dependency. Now Storage, not Media Services, keeps the information that belongs in Storage.

Where did client-side storage encryption go?

We now recommend server-side storage encryption (which is on by default). For more information, see [Azure Storage Service Encryption for Data at Rest](#).

What is the recommended upload method?

We recommend the use of HTTP(s) ingests. For more information, see [HTTP\(s\) ingest](#).

How does pagination work?

Media Services supports \$top for resources that support OData but the value passed to \$top must be less than 1000 (for example, the page size for pagination).

This allows you to either get a small sample of items using \$top (for example, the 100 most recent items) or to page through all items using pagination.

Media Services does not support paging through the data with a user specified page size.

For more information, see [Filtering, ordering, paging](#)

How to retrieve an entity in Media Services v3?

v3 is based on a unified API surface, which exposes both management and operations functionality built on **Azure Resource Manager**. In accordance with **Azure Resource Manager**, the resource names are always unique. Thus, you can use any unique identifier strings (for example, GUIDs) for your resource names.

Next steps

[Media Services v3 overview](#)

