Sitecore Media Framework 1.2

# Integration Guide

*A Developer's Guide to Building a Connector*

# Table of Contents

# Chapter 1

# Introduction

This chapter provides background information on Media Framework and an overview of the tasks involved in developing a custom connector.

## 1.1 Goals of this Document

This document was written with the following goals in mind:

- Enable developers to understand the components that make up a Media Framework connector.
- Provide technical guidance to developers, so that they can build their own Media Framework connectors.

## 1.2 Overview

This section provides background information on Media Framework.

### 1.2.1 Benefits to Content Authors

Video is an important part of a digital experience. Media Framework was designed to allow content authors to incorporate video into their digital channels in a consistent way, regardless of the video hosting provider.

Media Framework allows content authors to:

- Display different videos for different visitors based on their behavior.
- Recognize conversion for complete or partial video playback.
- View video playback activity as a part of the visitor's activity stream.
- Create, update and delete media within Sitecore Media Library.
- Embed video using a Sitecore rendering.
- Embed video using Sitecore rich text editor.
- Discover and use media using Sitecore standard tool set, such as Page Editor.

### 1.2.2 Benefits to Developers

Media Framework enables developers to integrate media service providers with Sitecore in a consistent manner.

Additionally, Media Framework has all the necessary components needed to enable content authors to incorporate media into their digital experience. Some examples include:

- Enable authors to embed video within rich-text fields using Sitecore content editor.
- Upload video from Sitecore content editor to external service provider.
- Provide video search result within the Sitecore client.
- Provide component renderings that embed video in pages.

By using Media Framework, a developer is able to focus his/her attention on the code needed to read from and write to an external media service instead of the code needed to integrate with Sitecore.

## 1.3 Architecture

While Media Framework is designed to make it as easy as possible to integrate with external media service providers, it does not mean that the integration process is trivial. Developer(s) must create a number of components before a fully-functional connector is implemented. These components include:

1. **Sitecore items** - various elements within Sitecore are responsible for representing external media, which allow content authors to manipulate external media within Sitecore. These elements includes: templates that represent external media, UI components that allow content authors to invoke commands like upload and cleanup, media provider account settings, video playback rules, etc.
2. **Media Importer -** the components responsible for retrieving media content from the external media service provider.
3. **Media Exporter -** the components responsible for updating media content on the external media service provider system.
4. **Media Uploader -** the component that uploads files to the external media service provider.
5. **Media Synchronizer -** the component that manages the synchronization between Sitecore items and the external media service provider content.
6. **Player Markup Generator -** the component that generates the HTML markup and JavaScript required by the external media service provider in order to display and playback video content.
7. **Event Triggers -** the component responsible for capturing video events that integrates with Sitecore Analytics.
8. **Cleanup Executer -** the component responsible for removing Sitecore items that no longer exist on the external media service provider system.

This document guides a developer through the process of implementing these components and configuring Sitecore to integrate with Media Framework.

## 1.4 Conventions Used in this Document

This document shows how to create a Media Framework connector for an imaginary vendor named "`MyCompany`". Anywhere the name "`MyCompany`" is presented, it must be substitute with the external media service provider's name. Similarly, all references to `<ServiceProvider>` must also be replaced with the provider's name (i.e. `MyCompany`).

# Chapter 2

# Sitecore Components

One of the Media Framework goals is to manage external media content within Sitecore. This requires that the external media is represented as Sitecore items, and enable Sitecore users to use and manipulate the media. This chapter covers:

- Creating data templates used to represent external media service accounts as well as various media content items.
- Configuring a branch template to facilitate the creation of new account hierarchy.
- Adding UI components (menu and toolbar) that allow content authors to interact with the external media service provider.

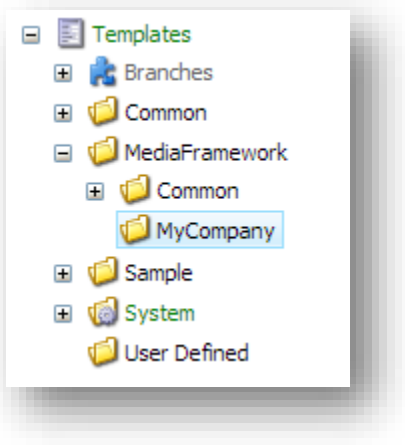
## 2.1 Data Templates

Media Framework connectors use Sitecore items to represent many elements, such as: service provider authentication credentials, video metadata, etc. This section describes the data templates that are included with Media Framework and outlines the steps for creating custom templates that are needed.

### 2.1.1 Create Templates Folder

A folder is needed to organize all of the custom data templates that support your connector.

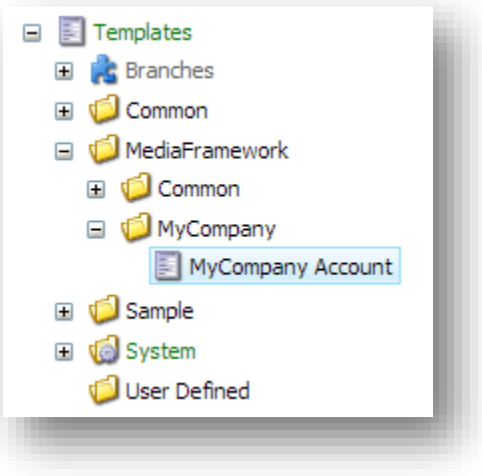
1. Open Content Editor.
2. Navigate to `/sitecore/templates/MediaFramework`.
3. Create a new item using the following settings:
    a. **Name:** MyCompany
    b. **Template:** `/sitecore/templates/System/Templates/Template Folder`



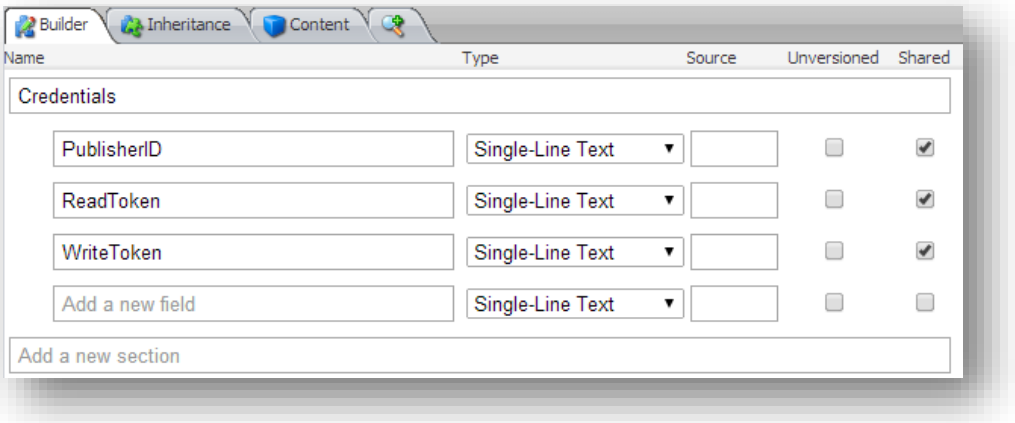

### 2.1.2 Create Account Template

The Account Template represents a user/vendor account on the external media service system. This template is used to store account credentials.

1. Create a new template using the following settings:
    a. **Name:** MyCompany Account
    **Base template:** /sitecore/templates/System/Templates/Standard template
    b. **Location:** /sitecore/templates/MediaFramework/MyCompany




2. Add fields that are necessary in order to identify a specific account on the external media service. The following screenshot is of the Account Template from the Brightcove connector.



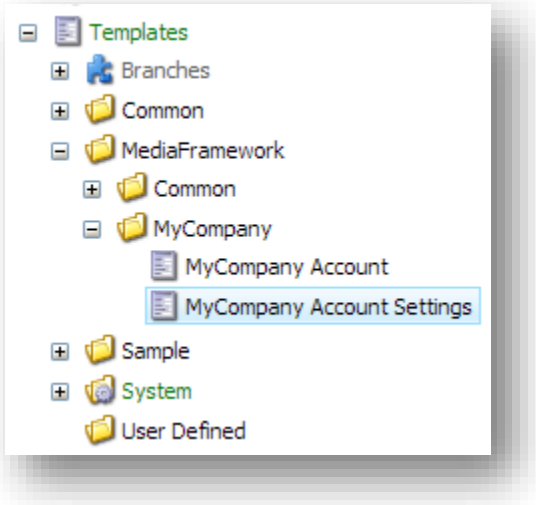

### 2.1.3 Create Account Settings Template

The Account Settings Template represents the settings that are configured on a per-account bases.

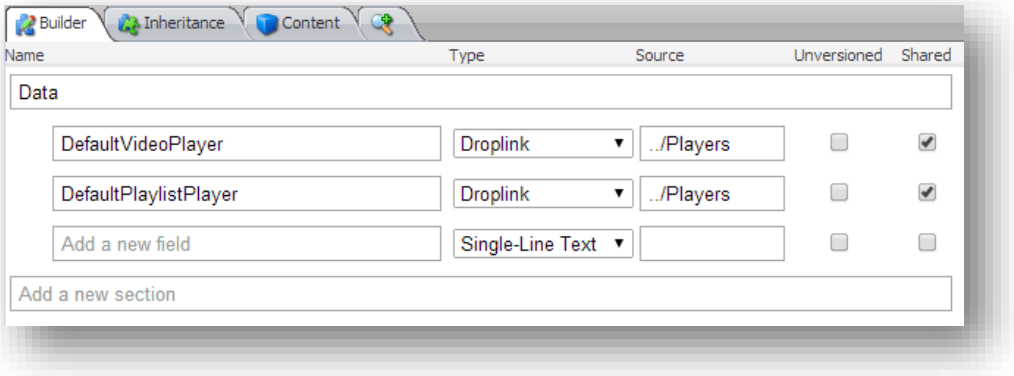
For example, the Brightcove connector uses account settings to define the default player for videos. This is not a setting that Brightcove itself supports. Rather, this is a setting that makes it easier to work with Brightcove within Sitecore.

Media Framework passes the account settings as a parameter through its API; therefore, this template must be created even if your connector do not require it.

1. Create a new template using the following settings:
    a. **Name:** MyAccount Account Settings
    b. **Base template:** `/sitecore/templates/MediaFramework/Common/AccountSettings`
    c. **Location:** `/sitecore/templates/MediaFramework/MyCompany`



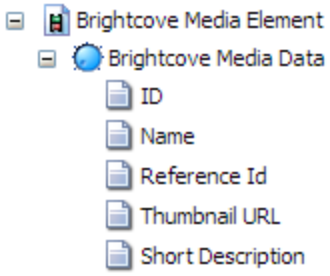

2. Add fields that are appropriate for your connector. The following screenshot is of the Account Settings Template from the Brightcove connector.

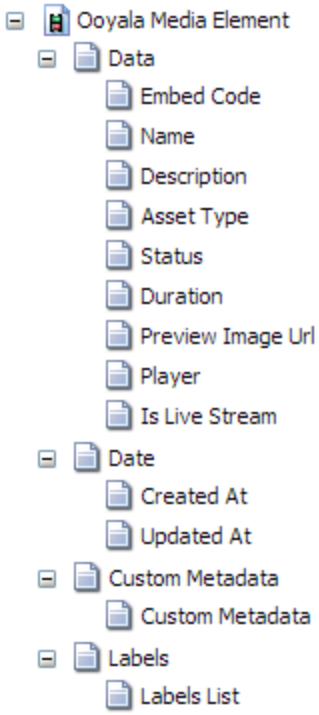



## 2.1.4 Create Media Element Template

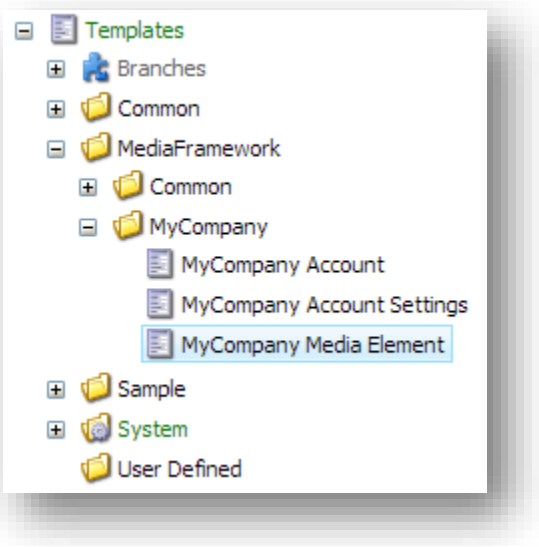
The Media Element Template is the base template for all custom media templates to inherit from. This template contains fields that are shared among all media elements. For example, the Brightcove connector uses a base template for videos and playlists items. The fields that are shared by both media items (such as "id", "name", and "short description") are defined in the Brightcove Media Element template.

The fields on this template depends on the data that needs to be captured from the external media provider. For a comparison, the following screenshots shows Brightcove connector and the Ooyala connector base Media Element Template.




1. Create a new template using the following settings:
    a. **Name:** MyCompany Media Element
    (for example, the template for the Brightcove connector is named "Brightcove Media Element")
    b. **Base template:**
    `/sitecore/templates/MediaFramework/Common/MediaElement`
    c. **Location:** `/sitecore/templates/MediaFramework/[templates folder]`




2. Add fields that are appropriate for your connector.

**Note:**
When determining which fields to add to the Media Element template, consider whether the field value needs to be stored within Sitecore.

Examples of fields to include in the Media Element template include: video ID (in order to uniquely identify the video), description (to allow a content author to edit the description without leaving Sitecore) and category (to allow a content author to search for a video).

**Note:**
When defining fields, consider how the field types affect content authors. For example, it might be faster to use a Single-Line Text field to store the keywords associated with a video; however, it makes it harder for the content author to edit keywords later. You may wish to look into how Brightcove and Ooyala connectors have implemented such fields.

### 2.1.5 Create Media Templates

For each media type that is exposed through the external service provider, a Sitecore data template is needed. Examples of media types include:

- Videos
- Playlists
- Players

Similar to Create Media Element Template (see section 2.1.4), the fields on these templates depend on the corresponding media types from the media service provider.

**Important**
Items that will be tracked via Sitecore analytics must be based on the Media Element template, such as: videos and playlists.

Items that will **not** be tracked via Sitecore analytics, should **not** be based on the Media Element template. An example of such an item is a video player.

## 2.2 Module Items

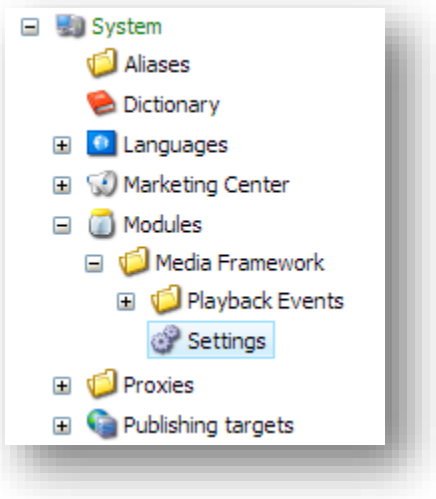
Media Framework is a single module that supports multiple connectors to different external media service providers. Each connector has its own settings. This section covers the creation of the items that represent connector specific settings.

### 2.2.1 Create Default Account Settings Item

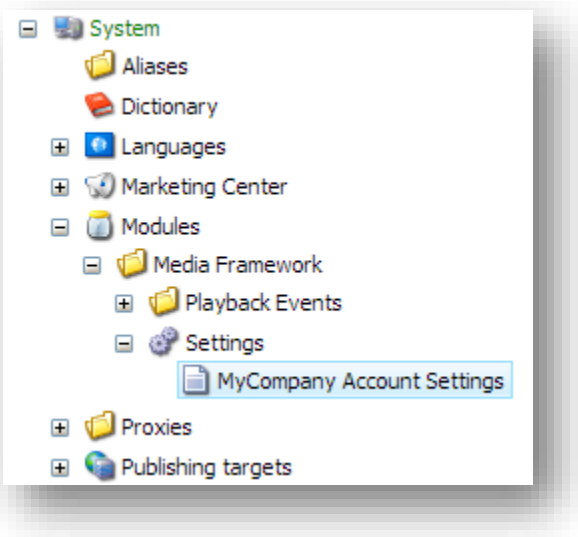
Media Framework supports multiple accounts per external media service provider. Each account can have its own default settings, which can be overwritten by a developer explicitly.

This section describes how to create an item that will store the default account settings.

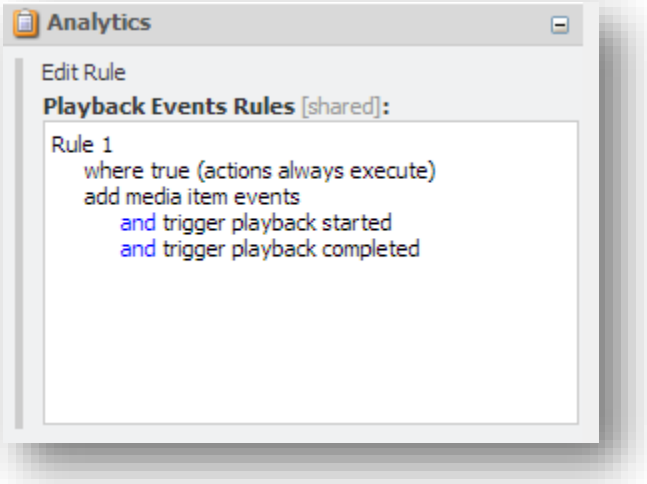
1. Open Content Editor.
2. Navigate to `/sitecore/system/Modules/Media Framework/Settings`




3. Create a new item using the following settings:
    a. **Name:** MyCompany Default Settings
    b. **Template:** `/MediaFramework/MyCompany/MyCompany Account Settings`




4. Set the following field value:
    a. **Field name:** Playback Events Rules
    b. **Value:** where true (actions always execute) add media item events and trigger playback started and trigger playback completed



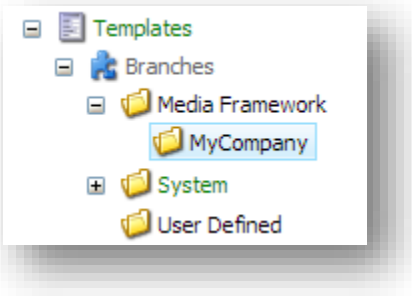
## 2.3 Branch Templates

When a developer adds a new Media Framework account to Sitecore, an account item is needed. This item stores the credentials for the account. Additionally, each account needs a hierarchy of media items, which are organized by the media type. For example, when a Brightcove account is created, child items are created to store media content, video players and tags. Branch templates ensure that the proper child items are created.

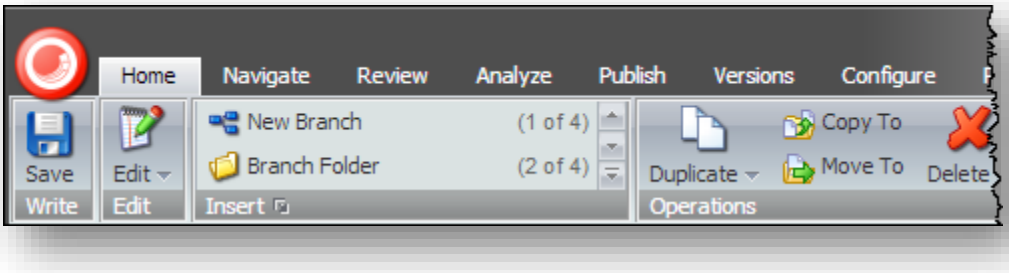
### 2.3.1 Create Account Branch Template

The branch template for an account will ensure that the correct child items are created when a new account item is created.

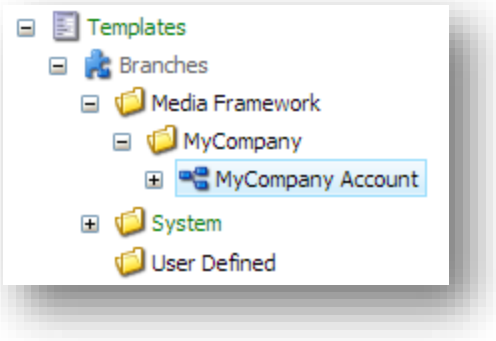
1. Open Content Editor.
2. Navigate to `/sitecore/templates/Branches/Media Framework`.
3. Create a new item using the following settings:
    a. **Name:** MyCompany
    b. **Template:** `/sitecore/templates/System/Branches/Branch Folder`



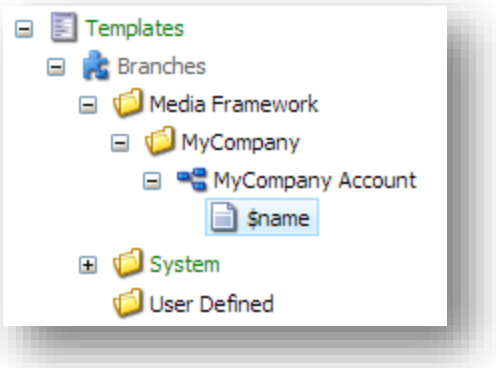

4. Navigate to the folder you just created.
5. Add a new item using the "New Branch" option. When prompted to select a template, select the account template you created in section 2.1.2.



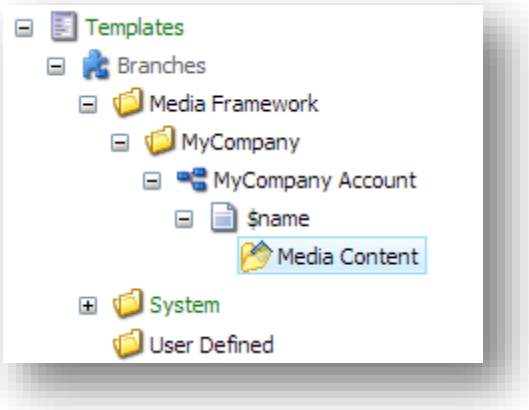

6. Create a new item using the following settings:
    a. **Name:** MyCompany Account
    b. **Template:** `/sitecore/templates/System/Branches/Branch`



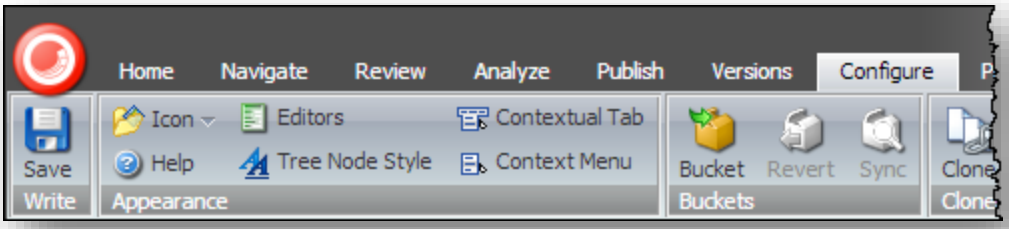

7. Expand the `MyCompany Account` item and select `$name`



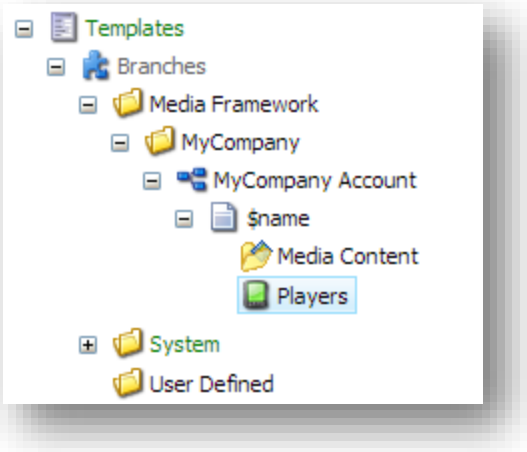

8. Add a new folder named "`Media Content`".
   Set the item icon to be `multimedia/32x32/open_image.png`.




9. Select the Media Content item.
10. In the ribbon click Configure > Bucket.



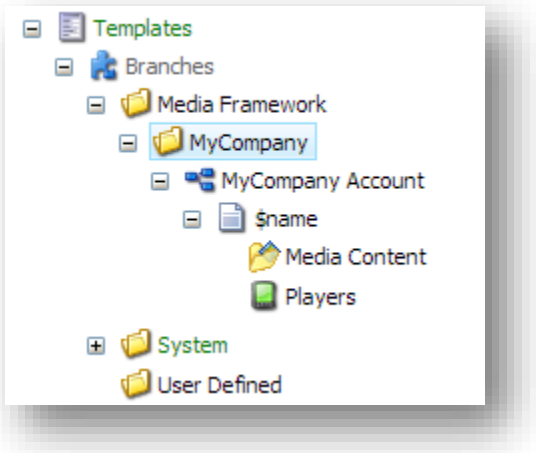

11. Add a new folder named "`Players`".
    Set the item icon to be `people/32x32/pda.png`.



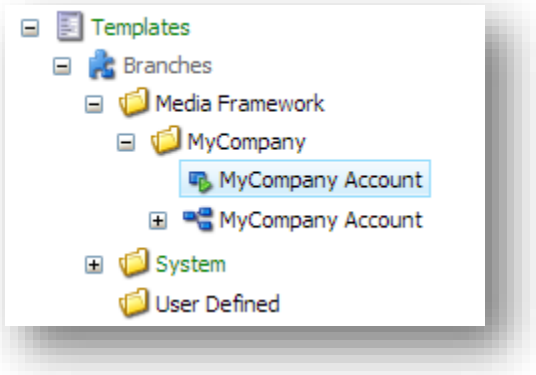

### 2.3.2 Create Account Command

When a new account is created, a command template is used. This ensures that the new account is configured properly in Media Framework

1. Navigate to /sitecore/templates/Branches/Media Framework/MyCompany/MyCompany Account
2. Copy the item ID. This will be referred to as "[ID 1]" later.
3. Navigate to /sitecore/system/Modules/Media Framework/Settings/MyCompany Account Settings
4. Copy the item ID. This will be referred to as "[ID 2]" later.
5. Navigate to /sitecore/templates/Branches/Media Framework/MyCompany

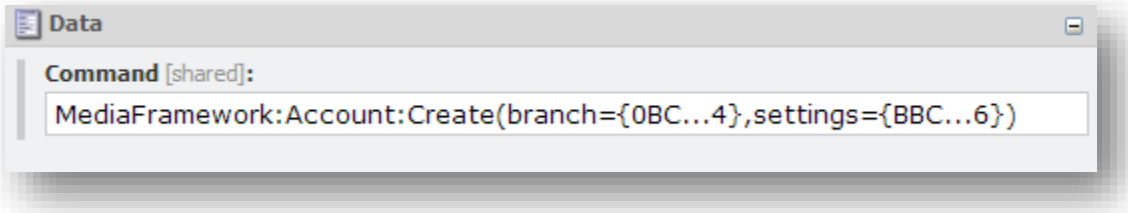



6. Create a new Command Template item with the name "`MyCompany Account`".




7. Set the value of the field "Command" to the following. Be sure to replace [ID 1] and [ID 2] with the item IDs noted previously:

```
MediaFramework:Account:Create(branch=[ID 1],settings=[ID 2])
```



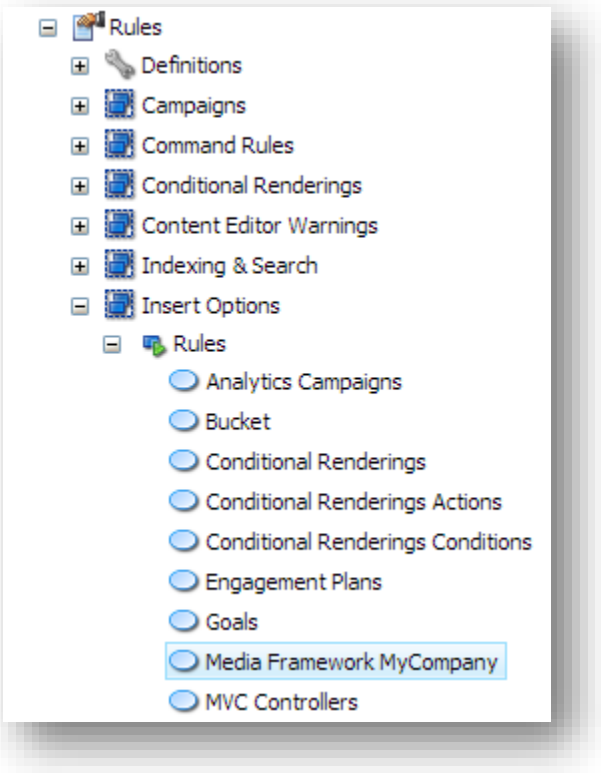
## 2.4 Insert Options

Insert options control where new items can be created. Users can create two types of items in Media Framework, which include: account items and media items. This section describes how to provide insert options for these two item types.

### 2.4.1 Create Insert Options Rule Item

Insert Options Rules provide a centralized way to control insert options.

1. Open Content Editor.
2. Navigate to `/sitecore/system/Settings/Rules/Insert Options/Rules`
3. Create a new item using the following settings:
    a. **Name:** Media Framework MyCompany
    b. **Template:** `/sitecore/templates/System/Rules/Insert Options Rule`

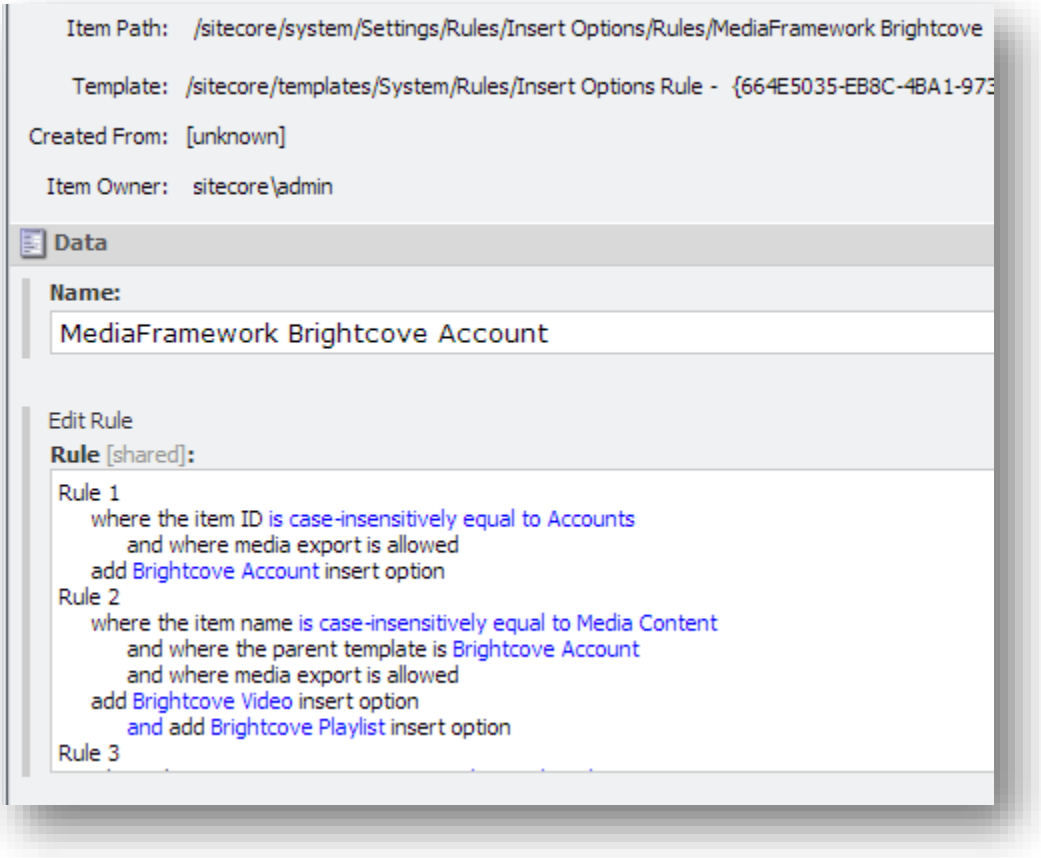



4. Set the following field value:
    a. **Field name:** Name
    b. **Value:** Media Framework MyCompany Account

### 2.4.2 Add Rule for Account Creation

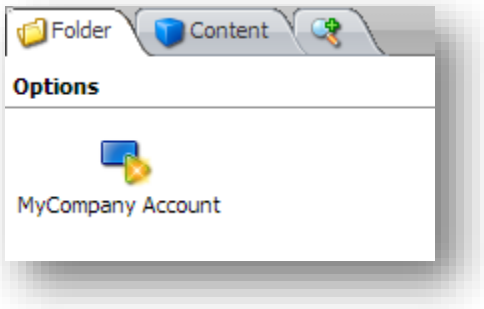
Media Framework accounts are defined in the Media Library. This section describes how to configure insert options so that your service provider Account items can be created.

1. Navigate to /sitecore/system/Settings/Rules/Insert Options/Rules/Media Framework `<ServiceProvider>`
2. Add appropriate insert options rules like the following:
    a. **Rule:** Where the item ID is case-insensitively equal to **Accounts** and where media export is allowed
    **Action:** add New `<ServiceProvider>` Account insert option
    b. **Rule:** Where the item ID is case-insensitively equal to **Accounts** and where the parent template is `<ServiceProvider>` Account and where media export is allowed
    **Action:** add New `<ServiceProvider>` Account insert option
    c. **Rule:** Where the item ID is case-insensitively equal to Players and where the parent template is `<ServiceProvider>` **Account** and where media export is allowed
    **Action:** add `<ServiceProvider>` Video Player insert option




**Note:**
The item ID must correspond to the item ID from the item `/sitecore/media library/Media Framework/Accounts`. The insert option uses the command template created in section 2.3.2.

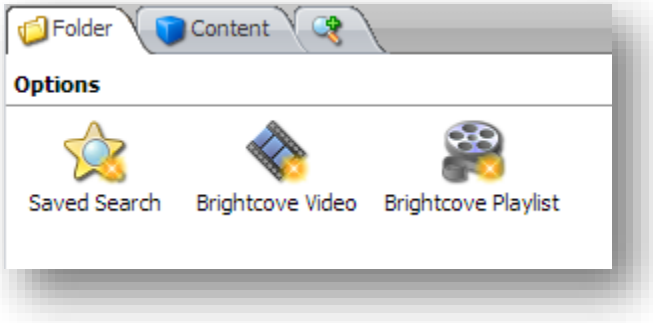
3. Now when you select `/sitecore/media library/Media Framework/Accounts` you should see the "MyCompany Account" command template as an available template.




### 2.4.3 Add Rules for Media Creation

If your Media Framework connector allows media items to be created in Sitecore, you must add additional rules to allow media items to be created under the item "Media Content".

The following is an example of the insert options rule for the Brightcove connector.



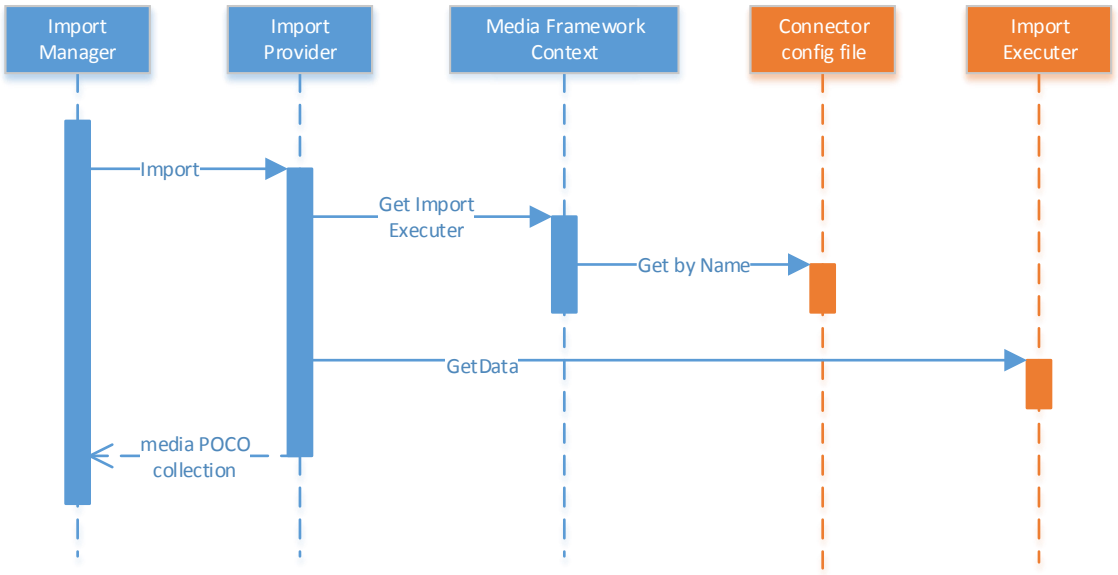
# Chapter 3

# Media Import

This chapter covers the media import functionality for a Media Framework connector.

## 3.1 Overview

The following sequence diagram illustrates the media import process. The orange boxes represent the components that must be implemented. The blue boxes are part of Media Framework, and do not need to be changed.



## 3.2 Import Executer

An import executer is responsible for retrieving media metadata from an external media service provider.

### 3.2.1 Identify Import Media Types

Developers must begin by identifying the types of media content needed for import. Media type that must be presented or managed through Sitecore are strong candidate for selection. These media types are contingent on what media service provider has made available through its API. Examples of typical media types include: video, playlists, players and tags.

### 3.2.2 Create Classes for Import Media Types

An import executer reads different types of media content from an external service provider and converts them into a collection of .NET objects. Therefore, there is a need for a .NET class for each type of media identified in section 3.2.1.

These objects are simple POCO objects that depend on the external service provider's API and the templates defined in section 2.1.4 and 2.1.5.

The following is an example of a class that represents a video item:

```
public class Video
{
  public DateTime CreationDate { get; set; }
  public string Id { get; set; }
  public DateTime LastModifiedDate { get; set; }
  public string Name { get; set; }
  public string Description { get; set; }
}
```

### 3.2.3 Implement IImportExecuter

The interface `Sitecore.MediaFramework.Import.IImportExecuter` has a single method that must be implemented. This method is responsible for retrieving a list of media content from an external service provider.

#### GetData()

| | | | |
|---|---|---|---|
| **Method:** | `IEnumerable<object> GetData(Item accountItem)` | | |
| **Description:** | Retrieves a collection of media content from an external service provider. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service provider. |
| **Return value** | | | |
| | **Type:** | `System.Collections.Generic.IEnumerable<object>` | |
| | **Description:** | Collection of media from the external media service | |

The following is an example of a class that implements the `IImportExecuter` interface:

```
public class VideoCollectionImporter : IImportExecuter
{
  public IEnumerable<Video> GetData(Item accountItem)
  {
    var list = new List<Video>();
    return list;
  }
}
```

### 3.2.4 Define Import Executers

The import provider reads import executers from the Sitecore configuration files. Import executers must be defined under the `configuration > sitecore > mediaFramework > mediaImport > importExecuters` section.

Each import executer has `name` and `type` attributes. The `name` attribute uniquely identifies the import executer among other importers. The `type` attribute is a fully-qualified type name for the class that implements the import executer.

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

The following is an example of the configuration for the import executer from section 3.2.3.

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <mediaFramework>
      <mediaImport>
        <importExecuters>
          <add name="import mycompany videos"
type="MyCompany.MediaFramework.Import.VideoCollectionImporter, MyCompany.MediaFramework"/>
        </importExecuters>
      </mediaImport>
    </mediaFramework>
  </sitecore>
</configuration>
```

### 3.2.5 Define Import Execution Scope

Scope executers are used to point to a list of executers for processing. For example, scheduling an import agent must import media contents like: videos, players, playlists, tags, etc. Scope executor provides a way to reference all importers using a unique identifier, so that it can be referenced via configuration file or Sitecore menu control.

Execution scopes are defined in Sitecore configuration files under `configuration > sitecore > mediaFramework > scopeExecuteConfigurations`.

Each execution scope is configured with the following:

- **Name** - used to identify the execution scope when the import process runs.
- **Type** - the class that represents the execution scope within the Media Framework API. The default class is `Sitecore.MediaFramework.Scopes.ScopeExecuteConfiguration`. It is unlikely you will need to change this class.
- **Account template** - the ID of the data template that represents an account for the external media service provider. This is the template that was created in section 2.1.2.

- **Import executers** - a list of import executers that should be run when the import process runs. The import executers should be listed in the order they need to run. For example, the import executer for videos would run before the import executer for playlists.

The following is an example of the configuration for the account template created in section 2.1.2 and the import executer from section 3.2.3:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
            <mediaFramework>
              ...
              <scopeExecuteConfigurations>
                <add name="import_mycompany_content"
type="Sitecore.MediaFramework.Scopes.ScopeExecuteConfiguration, Sitecore.MediaFramework">
                  <accountTemplate>{6F95D680-8B80-49AB-862A-3F83ACFE5615}</accountTemplate>
                  <scope hint="list">
                    <name>import_mycompany_videos</name>
                  </scope>
                </add>
              </scopeExecuteConfigurations>
            </mediaFramework>
          </sitecore>
        </configuration>
```

### 3.2.6 Define Import Schedule

Import scheduling agent must be specified in order to keep Sitecore database in sync with the external media provider system. A scheduling agent is defined under `configuration > sitecore > scheduling` section.

Scheduling agent is added using the `agent` node, which has three attributes: `name`, `type`, and `interval`. The `name` attribute is a unique name for the scheduling agent. The `type` attribute is the fully-qualified type name to the media framework import class, `Sitecore.MediaFramework.Scopes.ScopeExecuteConfiguration, Sitecore.MediaFramework`. The `interval` is the sleep duration between every execution.

The `agent` node has two additional child nodes to target Sitecore database and to determine the import scope. The database is specified using the `param` node. The import scope is specified by simply referencing the scope definition created in section 3.2.5 using xpath.

The following example illustrates the import agent configuration setting:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
           <sitecore>
              <scheduling>

                 <agent name="MediaFramework Import MyCompany"
                        interval="04:00:00"
                        type="Sitecore.MediaFramework.Schedulers.ImportScheduler,
Sitecore.MediaFramework">

                     <param desc="database">master</param>

                     <scopeConfigurations hint="raw:AddConfiguration">

                         <add ref="mediaFramework/scopeExecuteConfigurations
/*[@name='import_mycompany_content'][1]"/>

                     </scopeConfigurations>

                  </agent>


              </scheduling>
```

```
        </sitecore>
    </configuration>
```

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

## 3.3 Import Command

The import process can be started manually from Sitecore content editor. This section describes how to add this functionality for your connector.

### 3.3.1 Define Import Command

Media Framework includes a command that will trigger the import process. Since the command is invoked from within the Sitecore client, Sitecore knows which item is currently selected. The command execution is context sensitive and is able to determine appropriate account. The command must also include the execution scope.

Commands are defined in Sitecore configuration files under `configuration > commands`.

Each command has a `name` and a `type` attribute. The `name` attribute uniquely identifies the command. The `type` attribute is the fully-qualified type name for the class that implements the command.

Media Framework includes a command for handling the import process. Therefore, the `type` attribute should be set as `Sitecore.MediaFramework.Commands.ImportContent`. It is unlikely you will need to change this class.

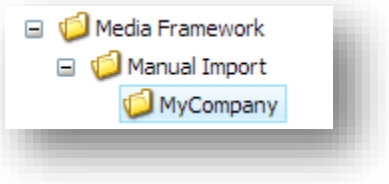
The following is an example of the configuration for the manual import command:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
            <commands>
              <command name="mediaFramework:ManualImport:MyCompany"
type="Sitecore.MediaFramework.Commands.ImportContent,Sitecore.MediaFramework"/>
            </commands>
            ...
          </sitecore>
        </configuration>
```

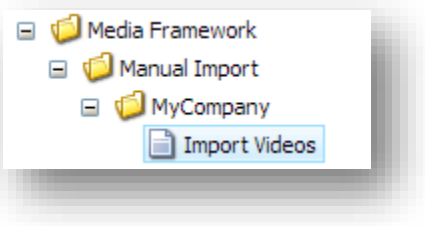
### 3.3.2 Add the Manual Import Button Menu Options

Media Framework provides the ability to import individual media types. For example, you can import just videos or just the playlists. The following section describes the steps needed to enable the import button.

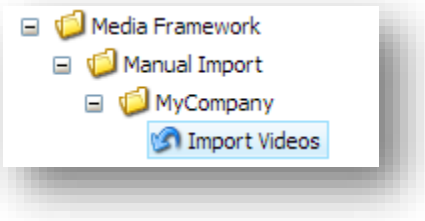
1. Open Content Editor.
2. Switch to the **core** database.
3. Navigate to `/sitecore/content/Applications/Content Editor/Menues/Media Framework/Manual Import`.
4. Create a new folder named "MyCompany".




5. Create a new item using the following settings:
    a. **Name:** Import Videos
    b. **Template:** `/sitecore/templates/System/Menus/Menu item`




6. Change the item icon to applications/24x24/undo.png

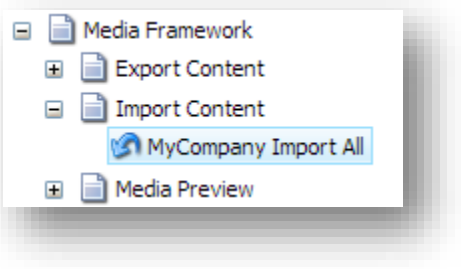



7. Set the following field value:
    a. **Field name:** Display name
    b. **Value:** Import Videos
8. Set the following field value:
    a. **Field name:** Icon
    b. **Value:** applications/24x24/undo.png
9. Set the following field value:
    a. **Field name:** Message
    b. **Value:** mediaFramework:ManualImport:MyCompany(scope=import_mycompany_videos)
10. Add additional "Menu item" items for each of the media types your import process supports.

### 3.3.3 Add the Manual Import Button

Media Framework provides the ability to import all media for an account. For example, you can import the videos and playlists with one command. The following steps show how to provide “Import All” functionality:

1. Open Content Editor.
2. Switch to the core database.
3. Navigate to `/sitecore/content/Applications/Content Editor/Ribbons/Chunks/Media Framework/Import Content`
4. Create a new item using the following settings:
    a. **Name:** MyCompany Import All
    b. **Template:** `/sitecore/templates/System/Ribbon/Large Menu Combo Button`




5. Set the following field value:
    a. **Field name:** Header
    b. **Value:** Import All
6. Set the following field value:
    a. **Field name:** Icon
    b. **Value:** applications/24x24/undo.png
7. Set the following field value:
    a. **Field name:** Click
    b. **Value:** mediaFramework:ManualImport:MyCompany(scope=import_mycompany_content)
8. Set the following field value:
    a. **Field name:** Tooltip
    b. **Value:** Import all content for the account
9. Set the following field value:
    a. **Field name:** Menu
    b. **Value:** `/sitecore/content/Applications/Content Editor/Menues/Media Framework/Manual Import/MyCompany`

## 3.4 Import Scheduler

The import process can be started automatically using a Sitecore scheduled agent. This section describes how to add this functionality to your connector.

### 3.4.1 Define the Import Agent

Media Framework includes an agent that is able to run the media import process. All you have to do is add the agent to your connector's configuration file.

The agent accepts the following parameters:

- **Database name** - the name of the Sitecore database where the imported media items should be stored.
- **Execution scopes** - the execution scope that the import agent should run.

The following is an example of the configuration for the import scheduling agent:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
            ...
            <scheduling>
              <agent name="MediaFramework Import MyCompany"
type="Sitecore.MediaFramework.Schedulers.ImportScheduler, Sitecore.MediaFramework"
interval="04:00:00">
                <param desc="database">master</param>
                <scopeConfigurations hint="raw:AddConfiguration">
                  <add
ref="mediaFramework/scopeExecuteConfigurations/*[@name='import mycompany content'][1]"/>
                </scopeConfigurations>
              </agent>
            </scheduling>
          </sitecore>
        </configuration>
```

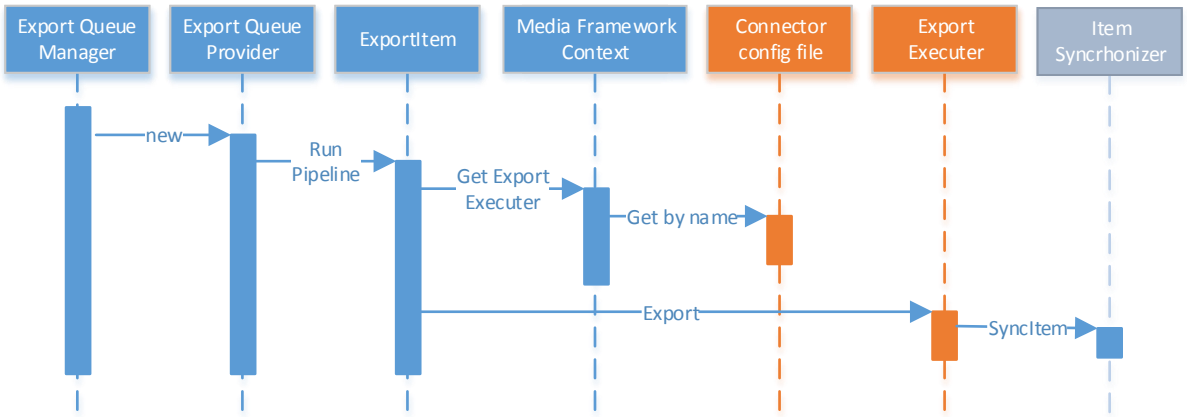
# Chapter 4

# Media Export

This chapter covers the media export functionality for a Media Framework connector.

## 4.1 Overview

The following sequence diagram illustrates the media export process. The orange boxes represent the components that must be implemented. The blue boxes are part of Media Framework, and do not need to be changed. The grey item, Media Synchronizer, is discussed in Chapter 5 of this document.



## 4.2 Export Executer

An export executer is responsible for updating media metadata on the external media service provider.

### 4.2.1 Identify Export Media Types

For each Import Executer (see Chapter 3), you'll likely need to create a respective export executers. Export executers enable content authors to update service provider media content from inside the Sitecore.

### 4.2.2 Reuse Classes for Exporting Media Types

You will need a .NET POCO class for each media type that will be exported. For example, a new entity instance must be returned after a create task is passed for an export operation. These entity classes are the same classes created for importers in section 3.2.2.

### 4.2.3 Implement ExportExecuterBase

Developer(s) must create exporter(s) that inherit from the abstract class `Sitecore.MediaFramework.Export.ExportExecuterBase`, which implements the interface `IExportExecuter`. The interface requires only three methods (IsNew, Export, and NeedToUpdate); however, the utilization of the abstract class provides the following benefits:

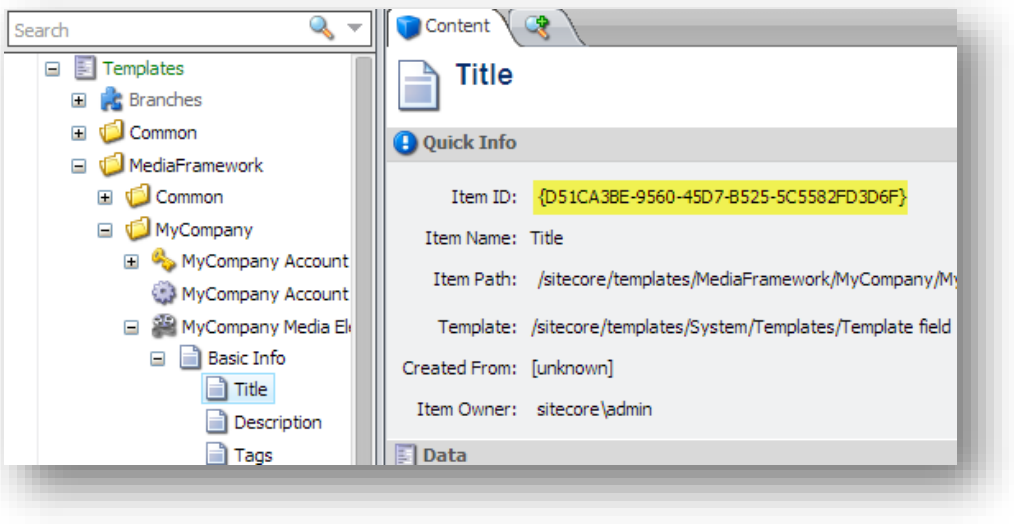
- Consistent implementation across media providers.
- Field list that is used to identify items to export.
- Explicit methods for Create, Update, Delete, and Move operation.
- Synchronization of dependent items (i.e. synchronize playlist after exporting a video item).

The following section describes the `ExportExecuterBase` property and methods:

#### FieldsToUpdate

| | | | |
|---|---|---|---|
| **Property:** | `protected abstract ID[] FieldsToUpdate { get; }` | | |
| **Description:** | This getter shall return a list of media data template field IDs that will be used to determine if an export operation is needed. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *N/A* | `N/A` | N/A |
| **Return value** | | | |
| | **Type:** | `Sitecore.Data.ID[]` | |
| | **Description:** | Array of template field Ids for the media content. | |

```
protected override Sitecore.Data.ID[] FieldsToUpdate
{
     get
     {
         return new ID[]
          { new ID("{D51CA3BE-9560-45D7-B525-5C5582FD3D6F}") /* Title field        */
          , new ID("{1528A723-FFB7-42AC-ACCB-FB472A4076D4}") /* Description field  */
          , new ID("{1B83DF17-DE85-49BB-A742-D199F0FDEA72}") /* MyReadOnlyId field */
     };
}
```




## IsNew()

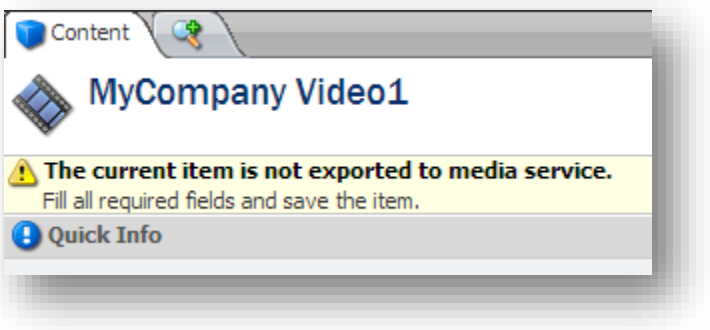
| | | |
|---|---|---|
| **Method:** | `public abstract bool IsNew (Item item);` | |
| **Description:** | This method returns true if the specified Sitecore data item has not been exported to the external service provider; otherwise, false is returned. When true, the content editor displays a warning message that the item has not been exported to an external system. To implement this item, you must create a **Read-Only** field in your data template declaration, which may only be set during synchronization process. | |
| **Parameters** | | |
| **Name** | **Type** | **Description** |
| *item* | `Sitecore.Data.Items.Item` | The Sitecore item that is to be exported. |
| **Return value** | | |
| **Type:** | `System.Boolean` | |
| **Description:** | The return value is true if Sitecore item is new and does not exist on the external media provider system. Otherwise, false is returned. | |

The following example, uses a read-only field to determine if the media item has been exported.

```
public override bool IsNew(Item item)
{
    var readOnlyFieldId = new ID("{1B83DF17-DE85-49BB-A742-D199F0FDEA72}");
    var readOnlyField = item.Fields[ readOnlyFieldId ];

    return readOnlyField == null
        || string.IsNullOrWhiteSpace( readOnlyField.Value );
}
```

When a new media item is not exported, the following warning message is shown in the content editor.

Figure 4.1




## Create()

| | | | |
|---|---|---|---|
| **Method:** | `protected abstract object Create(ExportOperation operation);` | | |
| **Description:** | This method is called after a new Sitecore item is created. The method returns an object that represents external media element. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *operation* | `Sitecore.MediaFramework.Export.ExportOperation` | Contains Sitecore media item, Sitecore item that represents the account to use when connecting to the external media service, and type of export operation (i.e. Create). |
| **Return value** | | | |
| | **Type:** | `object` | |
| | **Description:** | An instance of the media entity that represents the corresponding media type, identified in section 4.2.2. | |

The following example illustrates possible implementation:

```
        protected override object Create(ExportOperation operation)
        {
            var video = new MyCompany.Entities.Video()
            {
                FileName = operation.Item.Fields[FieldIds.FileName].Value,

                Updated = Sitecore.DateUtil.ParseDateTime(
                                operation.Item.Fields[FieldIds.Updated].Value,
                                DateTime.Now),

                Title = operation.Item.Fields[FieldIds.Title].Value,

                Description = item.Fields[FieldIds.Description].Value
                //...
            };

            // Same as:
            //   var synchronizer = new Sitecore.MediaFramework.MyCompany
            //                .Synchronize.EntityCreators.VideoEntityCreator();
            //   video = synchronizer.CreateEntity(operation.Item);


            //
```

```
            // Call External Media API
            //

            return video;
        }
```

## Delete()

| | | | |
|---|---|---|---|
| **Method:** | `protected abstract void Delete(ExportOperation operation);` | | |
| **Description:** | After a Sitecore item is deleted, this method is called to delete the external media content. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *operation* | `Sitecore.MediaFramework.Export.ExportOperation` | Contains Sitecore media item, Sitecore item that represents the account to use when connecting to the external media service, and type of export operation (i.e. Delete). |
| **Return value** | | | |
| | **Type:** | `void` | |
| | **Description:** | N/A | |

## Update()

| | | | |
|---|---|---|---|
| **Method:** | `protected abstract object Update(ExportOperation operation);` | | |
| **Description:** | After a Sitecore item is updated, this method is called to update the corresponding external media provider content. Similar to the create method, this method returns an object that represents the media element. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *operation* | `Sitecore.MediaFramework.Export.ExportOperation` | Contains Sitecore media item, Sitecore item that represents the account to use when connecting to the external media service, and type of export operation (i.e. Update). |
| **Return value** | | | |
| | **Type:** | `System.Object` | |
| | **Description:** | An instance of the media entity that represents the corresponding media type, identified in section 4.2.2. | |

## Move()

| | |
|---|---|
| **Method:** | `protected virtual object Move(ExportOperation operation)` |
| **Description:** | This method is called when a Sitecore item location path changes within the Sitecore content tree. |
| **Parameters** | |

| | Name | Type | Description |
|---|---|---|---|
| | *operation* | `Sitecore.MediaFramework.Export.ExportOperation` | Contains Sitecore media item, Sitecore item that represents the account to use when connecting to the external media service, and type of export operation (i.e. Move). |
| **Return value** | | | |
| | **Type:** | `System.Object` | |
| | **Description:** | An instance of the media entity that represents the corresponding media type, identified in section 4.2.2. | |

## NeedToUpdate()

| | | |
|---|---|---|
| **Method:** | `protected virtual bool NeedToUpdate(SaveArgs.SaveItem item)` | |
| **Description:** | This method is used to determine if any fields within a Sitecore item has changed during last save operation. The base class implementation compares every field value that was defined in `FieldsToUpdate` using the following expression: `field.get_Value() != field.get_OriginalValue()` | |
| **Parameters** | | |
| | **Name** | **Type** | **Description** |
| | *operation* | `SaveArgs.SaveItem` | Contains list of fields that must be checked to determine if an export operation is needed. |
| **Return value** | | | |
| | **Type:** | `System.Boolean` | |
| | **Description:** | True if there exists at least one field that has changed since last export operation. | |

## UpdateOnSitecore()

| | | |
|---|---|---|
| **Method:** | `protected virtual void UpdateOnSitecore(ExportOperation operation, object entity)` | |
| **Description:** | If there exists a synchronizer for an item, then respective synchronizer reference is called to handle inter-dependency between media item. For example, after updating/exporting video item, the respective playlist needs to be updated due to media provider implementation. Media Synchronizer is explained in Chapter 5 of this document. | |
| **Parameters** | | |
| | **Name** | **Type** | **Description** |
| | *operation* | `SaveArgs.SaveItem` | Contains list of fields that must be checked to determine if an export operation is needed. |
| | *entity* | `System.Object` | An instance of the media entity that represents the corresponding media type, identified in section 4.2.2. This is the entity instance that is returned from the |

| | | | |
|---|---|---|---|
| | | | earlier methods: Create, Update, and Move. |
| **Return value** | | | |
| | **Type:** | `void` | |
| | **Description:** | N/A | |

### 4.2.4 Define Export Executer

The export provider reads export executers from the Sitecore configuration files. Export executers must be defined under the `configuration > sitecore > mediaFramework > mediaExport > exportExecuters` section.

Each export executer is specified via `add` node, which has three attributes: `name`, `templateId`, and `type`. The `name` attribute uniquely identifies the export executer among other exporters. The `templateId` attribute is the media item templateId (created in section 2.1.5). Lastly, the `type` attribute is the fully-qualified type name for the class that implements the export executer.

Inside each exporter `add` node, there can exist zero or more synchronizer child nodes. The synchronizer inside the export references a synchronizer that is defined under `configuration > sitecore > mediaFramework > synchronizer.` The `synchronizer` contains `ref` attribute, which uses xpath expression to point to the `synchronizers` definition within the confugration file. Synchronizer definition is outlined in section 5.2.5.

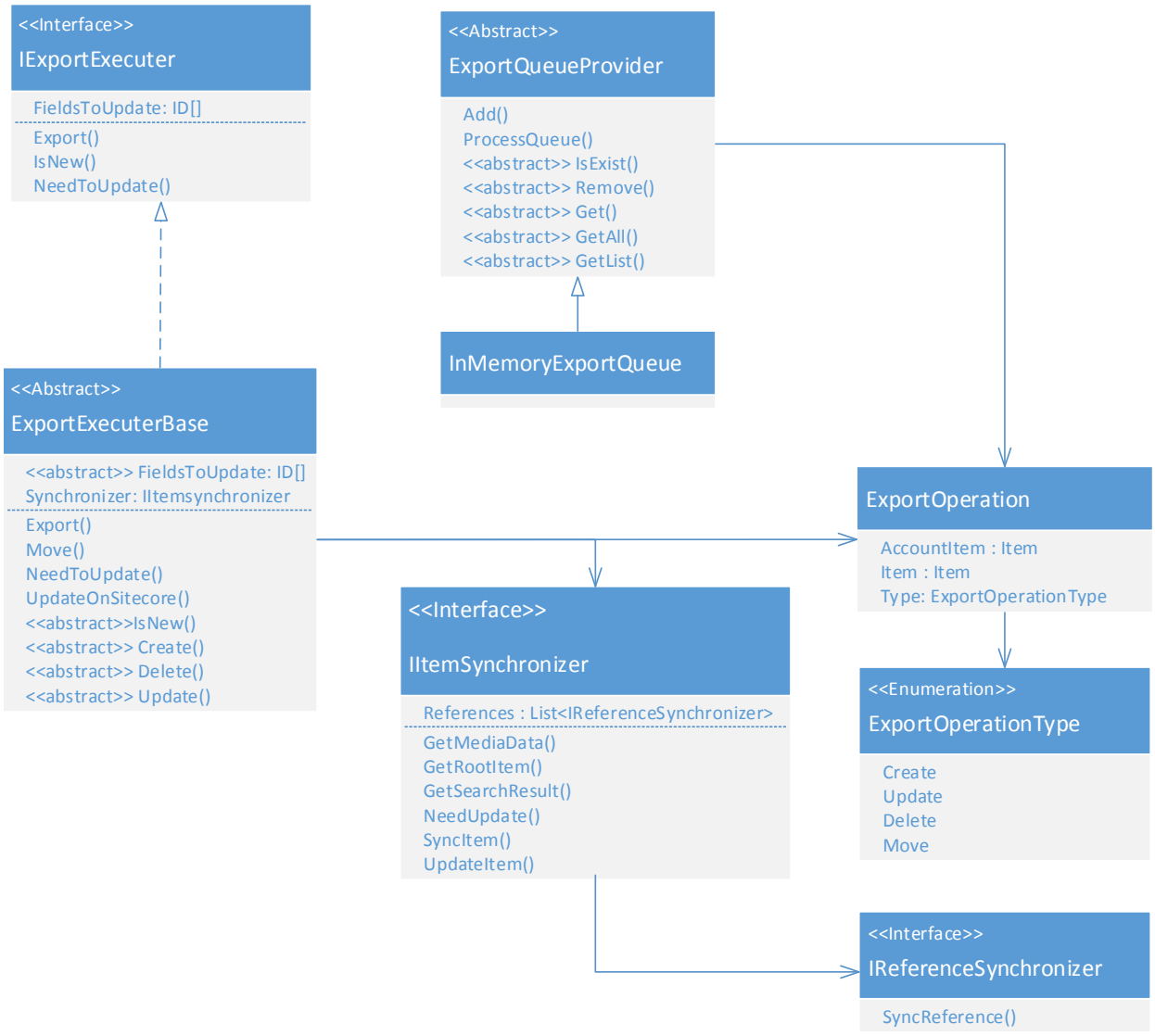
**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

The following is an example of the configuration for the export executer from section 4.2.3. Please note that the following illustration utilizes `synchronizer`, which is described in Chapter 5.

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
            <mediaFramework>

              <!-- ... -->

              <mediaExport>
                <exportExecuters>
                    <!—Ooyala exporter example:
                  <add name="ooyala video"
                       templateId="{C3B8C43D-AD6B-49E2-9C6E-2FB4B53A966A}"
type="Sitecore.MediaFramework.Ooyala.Export.VideoExporter, Sitecore.MediaFramework.Ooyala">
                    <synchronizer ref="mediaFramework/synchronizers/*[@name='ooyala_video'][1]"/>
                  </add>
                  -->

                  <add name="<ServiceProvider> <EntityName>"
                       templateId="<item template ID>"

type="Sitecore.MediaFramework.<ServiceProvider>.Export.<EntityName>Exporter,
Sitecore.MediaFramework.<ServiceProvider>">
                    <synchronizer
ref="mediaFramework/synchronizers/*[@name='<ServiceProvider>_<EntityName>'][1]"/>
                  </add>

                  <!-- ... -->

                </exportExecuters>
```

```
            </mediaExport>
          </mediaFramework>

          <!-- ... -->

        </sitecore>
      </configuration>
```

### 4.2.5 Class Diagram



## 4.3 Upload Executer

Creation of new media items from Sitecore is achieved by using an upload executer. Upload executer is responsible for uploading a media file to the external service provider, and then creating respective Sitecore item.

Sitecore upload UI form utilizes `Layouts/Upload.aspx` and jQuery upload plugin. It also uses Media Framework upload JavaScript, which is stored in `/sitecore modules/Web/MediaFramework/js/Uploader/MFUpload.js`.

### 4.3.1 Implement UploadExecuterBase

Developer(s) must create an upload executer that inherit from the abstract class `Sitecore.MediaFramework.Upload.UploadExecuterBase`, which implements the interface `IUploadExecuter`. This section describes the methods and properties for the `UploadExecuterBase`. Note that the `UploadInternal()` method is the only abstract method in this base class.

#### UploadInternal()

| | | | |
|---|---|---|---|
| **Method:** | `protected abstract object UploadInternal(NameValueCollection parameters, byte[] fileBytes, Item accountItem);` | | |
| **Description:** | This method is responsible for uploading a file to the external media service provider. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *parameters* | `System.Collections.Specialized.NameValueCollection` | Contains the collection of items that are specified in the constant class (read-only attributes) of `Sitecore.MediaFramework.Constants.Upload`. This collection includes: `RequestType, Cancel, Goto, UploadStatus, FileId, AccountId, MediaItemId, AccountTemplateId, FileName, Database, StartUpload, Progress, Id`. Because the collection values are string types, this abstract class provides helper methods that return respective strongly type data. |
| | *fileBytes* | `System.Byte[]` | Byte array of content (file) that needs to be uploaded. |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |

| | | |
|---|---|---|
| | **Type:** | `System.Object` |
| | **Description:** | Returns an instance of an entity that corresponds to the Sitecore item that is uploaded. The entity type is discussed in section 4.2.1 and 4.2.2.<br>If the upload is canceled, then an instance of `MediaFramework.Upload.CanceledVideo` is returned. |

The following example demonstrates the `UploadInternal()` implementation. It also shows the utilization of `UpdateStatus()` to provide upload feedback. Note, `ChunkSize` and `PostProcessingStatus` properties are set from the configuration file (see section 4.4).

```
public class VideoUploader : UploadExecuterBase
{
    public int ChunkSize { get; set; }  // Assigned from config file
    public string PostProcessingStatus { get; set; }  // Assigned from config file

    protected override object UploadInternal( NameValueCollection parameters
                                            , byte[] fileBytes, Item accountItem)
    {
        MyCompany.Entities.Video video = ExternalMediaProviderCreateVideoPlaceHolder(
                                         this.GetFileName(parameters));

        int hundredMeg = 104857600;
        int chunkSize = Math.Max(hundredMeg, fileBytes.Length / 100);

        bool isThereMoreToUpload = !this.IsCanceled(parameters)
                                && fileBytes.Length > 0;

        int startOffset = 0;

        while (isThereMoreToUpload)
        {
            int numberOfBytesLeft = fileBytes.Length - startOffset;

            int numberOfBytesToUpload = Math.Min(chunkSize, numberOfBytesLeft);

            try
            {
                ExternalMediaProviderSendData(fileBytes,
                                              startOffset, numberOfBytesToUpload);
            }
            catch (Exception e)
            {
                Sitecore.Diagnostics.Log.Error("Upload process failed!", e, this);

                UpdateStatus(mediaItemId: Guid.Empty,
                             fileId   : this.GetFieldId(parameters),
                             accountId: accountItem.ID.Guid,
                             progress : 0,
                             error    : "Upload process failed. Please see logs files!");

                return null;
            }

            startOffset += numberOfBytesToUpload;

            byte percentCompleted = System.Convert.ToByte((int)
                                    ((float)startOffset / fileBytes.Length * 100));

            UpdateStatus(mediaItemId: Guid.Empty,
                         fileId     : this.GetFieldId(parameters),
                         accountId  : this.GetAccountId(parameters),
                         progress   : percentCompleted);

            isThereMoreToUpload = !this.IsCanceled(parameters)
                                && startOffset < fileBytes.Length-1;
```

```
                } // while

                if (this.IsCanceled(parameters))
                {
                    Cancel(this.GetFieldId(parameters), accountItem.ID.Guid);
                    return new CanceledVideo();
                }

                return video;
            }
        //...
        }
```

## FileExtensions

| | | |
|---|---|---|
| **Property:** | `Public List<string> FileExtensions { get; }` | |
| **Description:** | This getter returns a list of file extension names prefixed with a dot (i.e. .avi, .mpeg, etc), which satisfies one of `IUploadExecuter` methods. Internally, this getter utilizes `protected List<string> fileExtensions` if available (not null); otherwise, it uses comma or pipe delimited values in `Extensions` (see `Extensions` property). This getter is used in `ValidateFileExtension()` method, which is called by the `Upload()` method. | |
| **Parameters** | | |
| **Name** | **Type** | **Description** |
| *N/A* | `N/A` | N/A |
| **Return value** | | |
| **Type:** | `System.Collections.Generic.List<string>` | |
| **Description:** | File inclusion list, which contains file extensions without the dot (.) prefix. | |

## Extensions

| | | |
|---|---|---|
| **Property:** | `public string Extensions { get; set; }` | |
| **Description:** | This property represents comma delimited values of file extensions, which is set through the configuration file. | |
| **Parameters** | | |
| **Name** | **Type** | **Description** |
| *N/A* | `N/A` | N/A |
| **Return value** | | |
| **Type:** | `System.String` | |
| **Description:** | Comma delimited supported file extensions. | |

Below configuration provides an example for setting the supported file extension.

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
            <mediaFramework>
                <mediaExport>
                  <uploadExecuters>

                    <add name="mycompany video"
                         accountTemplate="{1E5A076B-6A9A-4825-9FB1-3E95645352E7}"
                         type="Sitecore.MediaFramework.MyCompany.Upload.VideoUploader,
Sitecore.MediaFramework.MyCompany">
```

```
                        <synchronizer
ref="mediaFramework/synchronizers/*[@name='mycompany video'][1]"/>

                        <chunkSize>5000000</chunkSize>

<Extensions>wmv,avi,mov,moov,mpg,mpeg,m2t,m2v,vob,flv,mp4,mpg4,mkv,asf,m4v,m2p,3gp,3g2,f4v,mp3,m4
a,wma,aac</Extensions>

                        <postProcessingStatus>live</postProcessingStatus>
                    </add>
                </uploadExecuters>
            </mediaExport>
        </mediaFramework>
    </sitecore>
</configuration>
```

## fileExtensions

| | | | |
|---|---|---|---|
| **Property:** | `protected List<string> fileExtensions` | | |
| **Description:** | Instance variable to support `FileExtensions` property. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *N/A* | `N/A` | N/A |
| **Return value** | | | |
| | **Type:** | `System.Collections.Generic.List<string>` | |
| | **Description:** | See `FileExtensions` property. | |

## SupportCanceling()

| | | | |
|---|---|---|---|
| **Property:** | `Public bool SupportCanceling { get; set; }` | | |
| **Description:** | Used to determine if an upload process can be canceled. This methods complements the | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *N/A* | `N/A` | N/A |
| **Return value** | | | |
| | **Type:** | `System.Collections.Generic.List<string>` | |
| | **Description:** | Returns true if upload process can be canceled by the user; otherwise, false is returned. | |

## IsCanceled()

| | |
|---|---|
| **Method:** | `protected virtual bool IsCanceled(NameValueCollection parameters)` |
| **Description:** | This method complements the `SupportCanceling()` property and the `Cancel()` method, where the uploader can check if the operation is canceled so that no further data is transmitted. While uploading a file in chunks (see `UploadInternal()`) method, the developer must inspect the `IsCanceled()` method during every iteration of chunk upload; if true, then operation must be aborted and `Cancel()` method must be called. See `UploadInternal()` for an example. |
| **Parameters** | |

| | Name | Type | Description |
|---|---|---|---|
| | *parameters* | `System.Collections.Specialized.NameValueCollection` | Contains the collection of items that are specified in the constant class (read-only attributes) of `Sitecore.MediaFramework.Constants.Upload`. This collection includes: `RequestType, Cancel, Goto, UploadStatus, FileId, AccountId, MediaItemId, AccountTemplateId, FileName, Database, StartUpload, Progress, Id.` Because the collection values are string types, this abstract class provides helper methods that return respective strongly type data. |
| **Return value** | | | |
| | **Type:** | `System.Boolean` | |
| | **Description:** | True when the user or system has opted to cancel the upload process. False value denotes that the upload may continue without any interruption. | |

## Cancel()

| | | |
|---|---|---|
| **Method:** | `Public void Cancel(Guid fileId, Guid accountId)` | |
| **Description:** | Cancels the in-progress upload operation. This method satisfies one of the `IUploadExecuter` methods. Internally, it calls the static method `Sitecore.MediaFramework.Upload.UploadManger.Cancel()` which is specified in `Sitecore.MediaFramework.config` file under the following path `configuration > Sitecore > mediaFramework > uploadManager > providers.` | |
| **Parameters** | | |
| | **Name** | **Type** | **Description** |
| | *fileId* | `System.Guid` | See GetFileId`()` method |
| | *accountId* | `System.Guid` | See GetAccountId`()` method |
| **Return value** | | |
| | **Type:** | `void` |
| | **Description:** | N/A |

## Upload()

| | |
|---|---|
| **Method:** | `public void Upload(NameValueCollection parameters, byte[] fileBytes)` |

| | | | |
|---|---|---|---|
| **Description:** | This method is responsible for upload operations to the external media service provider. This method satisfies one of the interface `IUploadExecuter` methods, and calls `UploadInternal()` method mentioned earlier. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *parameters* | `System.Collections.Specialized.NameValueCollection` | Contains the collection of items that are specified in the constant class (read-only attributes) of `Sitecore.MediaFramework.Constants.Upload`. This collection includes: `RequestType, Cancel, Goto, UploadStatus, FileId, AccountId, MediaItemId, AccountTemplateId, FileName, Database, StartUpload, Progress, Id.` Because the collection values are string types, this abstract class provides helper methods that return respective strongly type data. |
| | *fileBytes* | `System.Bytes[]` | Byte array of content (file) that needs to be uploaded. |
| **Return value** | | | |
| | **Type:** | `void` | |
| | **Description:** | N/A | |

## ValidateFileExtension()

| | | | |
|---|---|---|---|
| **Method:** | `public virtual bool ValidateFileExtension(string fileName)` | | |
| **Description:** | Prior to uploading a file, its extension is checked in the `Upload()` method before initiating the upload via the abstract `UploadInternal()` method. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *filename* | `System.String` | |
| **Return value** | | | |
| | **Type:** | `System.Boolean` | |
| | **Description:** | True if the extension of `fileName` parameter is in the `FileExtensions` list; otherwise, false is returned. | |

## Synchronizer

| | |
|---|---|
| **Property:** | `public IItemSynchronizer Synchronizer { get; set; }` |
| **Description:** | This is the entity synchronizer (see Chapter 5) that insures Sitecore items are synchronized after export and upload operation. The `Synchronizer.SyncItem()` |

| | | | |
|---|---|---|---|
| | | method is called from the `Upload()`, after calling the abstract method `UploadInternal()`. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *N/A* | `N/A` | N/A |
| **Return value** | | | |
| | **Type:** | `Sitecore.MediaFramework.Synchronize.IItemSynchronizer` | |
| | **Description:** | See Chapter 5 for a description of Item Synchronizer. | |

## GetAccountItem()

| | | | |
|---|---|---|---|
| | **Method:** | `protected virtual Item GetAccountItem(NameValueCollection parameters)` | |
| | **Description:** | This helper function uses the key `MediaFramework.Constants.Upload.AccountId` to retrieve strongly typed Sitecore media account created by user. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *parameters* | `System.Collections.Specialized.NameValueCollection` | Contains the collection of items that are specified in the constant class (read-only attributes) of `Sitecore.MediaFramework.Constants.Upload`. Return respective strongly type data using name/value pair item. |
| **Return value** | | | |
| | **Type:** | `Sitecore.Data.Items.Item` | |
| | **Description:** | The Sitecore item that represents the account to use when connecting to the external media service provider. | |

## GetAccountId()

| | | | |
|---|---|---|---|
| | **Method:** | `protected virtual Guid GetAccountId(NameValueCollection parameters)` | |
| | **Description:** | This helper function uses the key `MediaFramework.Constants.Upload.AccountId` to retrieve strongly typed item GUID for the Sitecore media account created by user. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *parameters* | `System.Collections.Specialized.NameValueCollection` | Contains the collection of items that are specified in the constant class (read-only attributes) of `Sitecore.MediaFramework.Constants.Upload`. Return respective strongly type data using name/value pair item. |

| Return value | | |
|---|---|---|
| | **Type:** | `Sitecore.Data.ID` |
| | **Description:** | The Sitecore item ID that represents the account to use when connecting to the external media service provider. |

## GetDatabase()

| **Method:** | `protected virtual string GetDatabase(NameValueCollection parameters)` | |
|---|---|---|
| **Description:** | This helper function uses the key `MediaFramework.Constants.Upload.Database` to retrieve the context database name. | |
| **Parameters** | | |
| **Name** | **Type** | **Description** |
| *parameters* | `System.Collections.Specialized.NameValueCollection` | Contains the collection of items that are specified in the constant class (read-only attributes) of `Sitecore.MediaFramework.Constants.Upload`. Return respective strongly type data using name/value pair item. |
| **Return value** | | |
| **Type:** | `string` | |
| **Description:** | Context database name. | |

## GetFileName()

| **Method:** | `protected virtual string GetFileName(NameValueCollection parameters)` | |
|---|---|---|
| **Description:** | This helper function uses the key `MediaFramework.Constants.Upload.FileName` to retrieve the filename for media item. | |
| **Parameters** | | |
| **Name** | **Type** | **Description** |
| *parameters* | `System.Collections.Specialized.NameValueCollection` | Contains the collection of items that are specified in the constant class (read-only attributes) of `Sitecore.MediaFramework.Constants.Upload`. Return respective strongly type data using name/value pair item. |
| **Return value** | | |
| **Type:** | `System.String` | |
| **Description:** | Filename that is currently being uploaded. | |

## GetFileId()

| | | | |
|---|---|---|---|
| **Method:** | `protected virtual Guid GetFileId(NameValueCollection parameters)` | | |
| **Description:** | This helper function uses the key `MediaFramework.Constants.Upload.FileId` to retrieve strongly typed unique identifier, which shall be different during each upload operation. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *parameters* | `System.Collections.Specialized.NameValueCollection` | Contains the collection of items that are specified in the constant class (read-only attributes) of `Sitecore.MediaFramework.Constants.Upload`. Return respective strongly type data using name/value pair item. |
| **Return value** | | | |
| | **Type:** | `System.Guid` | |
| | **Description:** | Guid Id represents a unique identifier for the file. During each upload a different file Id is assigned, even if the same file is being uploaded. | |

**Note**
At the time of writing this document, `GetFileId()` method does not exist. Instead, `GetFieldId()` method is used; though the contextual meaning is different (due to typographical error), its function is to return a `FileId`.

## UpdateStatus()

| | | | |
|---|---|---|---|
| **Method:** | `protected virtual void UpdateStatus(Guid mediaItemId, Guid fileId, Guid accountId, byte progress, string error = null)` | | |
| **Description:** | Provides a visual feedback on the status of the media item upload. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *mediaItemId* | `System.Guid` | The Sitecore item that the update is for. The `Upload()` method of the abstract class `UploadExecuterBase` class sets this item after `Synchronizer.SyncItem();` otherwise, `mediaItemId` is set to `Guid.Empty`. |
| | *fileId* | `System.Guid` | Unique file identifier, which corresponds to the method `GetFileId()`. |
| | *accountId* | `System.Guid` | Item GUID for the Sitecore media account created by |

| | | | |
|---|---|---|---|
| | | | user, which corresponds to `GetAccountId()` method. |
| | *Progress* | `System.Byte` | Percentage of upload progress that is represented as a whole number from 0 to 100. |
| | *error* | `System.String` | Error message to display during upload (if any); otherwise, it defaults to null. |
| **Return value** | | | |
| | **Type:** | `Void` | |
| | **Description:** | N/A | |

## 4.4 Define UploadExecuterBase

The upload provider reads executers from the Sitecore configuration files. Upload executer must be defined under the `configuration > sitecore > mediaFramework > mediaExport > uploadExecuters` section.

Upload executers are specified using the `add` node, which has three attributes: `name`, `accountTemplate`, and `type`. The `name` attribute is the string name that uniquely identifies the upload executer among other uploaders. The `accountTemplate` attribute is the template Guid ID used to create the service provider account credentials. The `type` attribute is the fully-qualified type name for the class that implements the `UploadExecuterBase`.
The `add` node also has the `synchronizer` and the `Extensions` child nodes. The synchronizer specifies the `synchronizer` definition (see section 5.2.5) via xpath and the `Extensions` specifies supported media file extensions.

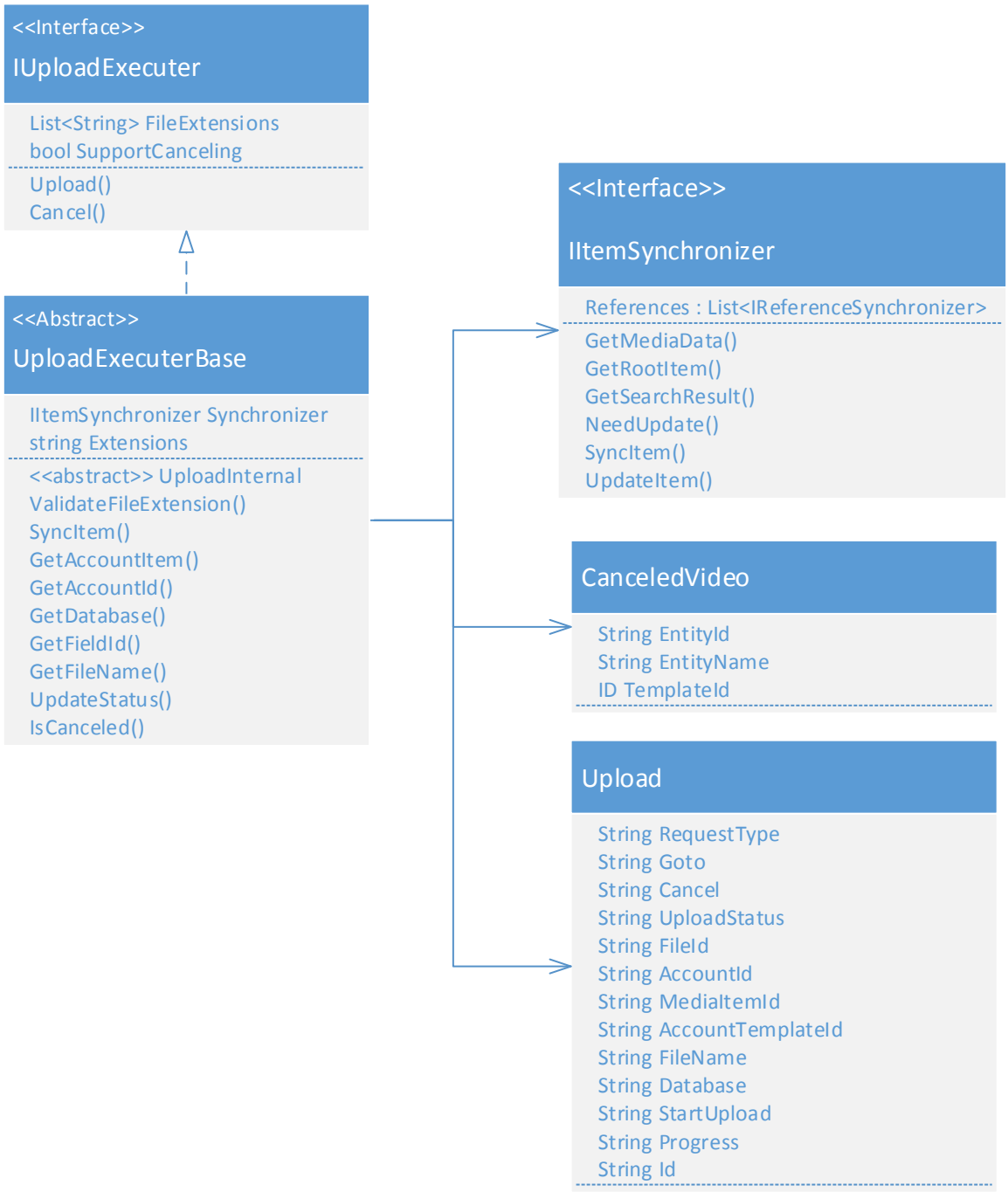
**Note:**
Extension node may not include whitespace between delimiters.

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

In the following upload executer configuration example, the `chunkSize` and `postProcessingStatus` nodes are not part of `UploadExecuterBase` properties; however, Sitecore will assign any child node to the respective property. For example, the property `MyCompanyUploadExecuter.ChunkSize` will be set to `5000000` for the following configuration setting.

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
            <mediaFramework>
                <mediaExport>
                  <uploadExecuters>

                    <add name="mycompany video"
                         accountTemplate="{1E5A076B-6A9A-4825-9FB1-3E95645352E7}"
                         type="Sitecore.MediaFramework.MyCompany.Upload.VideoUploader,
Sitecore.MediaFramework.MyCompany">

                       <synchronizer
ref="mediaFramework/synchronizers/*[@name='mycompany video'][1]"/>

                       <chunkSize>5000000</chunkSize>

<Extensions>wmv,avi,mov,moov,mpg,mpeg,m2t,m2v,vob,flv,mp4,mpg4,mkv,asf,m4v,m2p,3gp,3g2,f4v,mp3,m4
a,wma,aac</Extensions>

                       <postProcessingStatus>live</postProcessingStatus>
                    </add>
                  </uploadExecuters>
                </mediaExport>
            </mediaFramework>
          </sitecore>
        </configuration>
```

## 4.5 Class Diagram



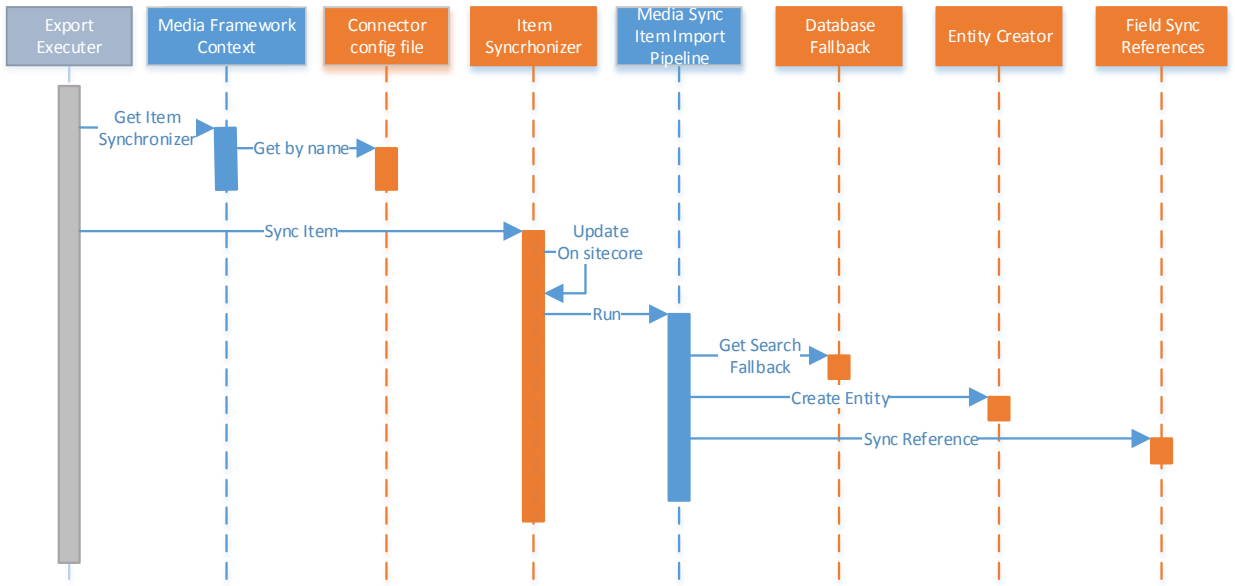
# Chapter 5

# Media Synchronizer

This chapter covers the media synchronization functionality for a Media Framework connector.

## 5.1 Overview

The following sequence diagram illustrates the synchronization process. The orange boxes represent the components that must be implemented. The blue boxes are part of Media Framework, and do not need to be changed. The grey item (Media Export) is discussed in Chapter 4 of this document.




Synchronizers are responsible for:

- Comparing Sitecore items with data retrieved from media service via Import Executer (Chapter 3).
- Update Sitecore items when imported items differ from Sitecore items.
- Update media service provider via Export Executer (Chapter 4) when Sitecore items are changed.
- Import interdependent items after export (save) operation. For instance, exporting a video item may affect video player that contains a playlist of items.

The implementation of synchronizers require the following components:

- Synchronizer executers – responsible for the synchronization process.
- Entity creator – Provides an instance of media entity for a given Sitecore item.
- Database Fallback – Provides search fallback capability, in the event that existing search implementation yield to empty result set (i.e. due to search engine index rebuild scheduling)
- Field References – Field references are used to check the dependency between exported items to external service provider and other Sitecore items that represent different media elements.

The sections that follow, elaborate on each point separately.

## 5.2 Item Synchronizer

A synchronizer is responsible for updating Sitecore media item.

### 5.2.1 Implement SynchronizerBase

Developer(s) must create synchronizers(s) that inherit from the abstract class `Sitecore.MediaFramework.Synchronize.SynchronizerBase`, which implements the interface `IItemSynchronizer`. The interface `IItemSynchronizer` inherits from `IDatabaseFallback`, and `IMediaServiceEntityCreator`. The implementation of these to interfaces is discussed later in this chapter.

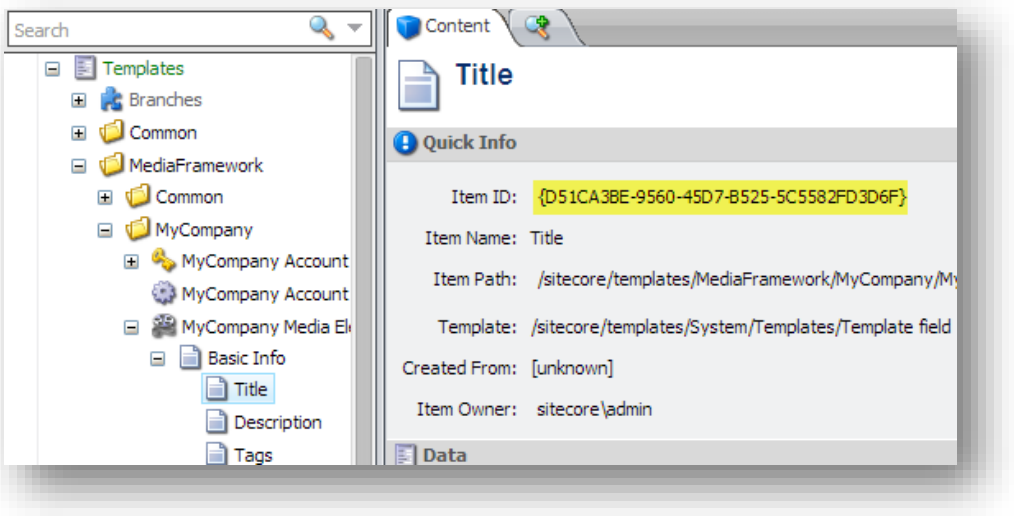
The following section describes the `SynchronizerBase` property and methods:

#### GetMediaData()

| | | | |
|---|---|---|---|
| **Method:** | `public abstract MediaServiceEntityData GetMediaData(object entity);` | | |
| **Description:** | Returns an instance of `MediaServiceEntityData` object for a given medial element, which is the minimum element needed to create a Sitecore item. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *Entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2. |
| **Return value** | | | |
| | **Type:** | `Sitecore.MediaFramework.Entities.MediaServiceEntityData` | |
| | **Description:** | `MediaServiceEntityData` parameter contains three properties, which include: `EntityId`, `EntityName`, and `TemplateId`. The `EntityId` property is recognized as a unique Id by the external service provider. Similarly, `EntityName` is an appropriate label recognized by the media provider. Lastly, `TemplateId` is the data template ID for the respective media type. | |

In below example, `MyReadOnlyId` variable is the service provider's unique identifier for a video item.

```
public override MediaServiceEntityData GetMediaData(object entity)
{
   // A single element that represents Media object
   var video = entity as MyCompany.Entities.Video;

   return new MediaServiceEntityData
   {
      EntityId = video.MyReadOnlyId,

      EntityName = video.FileName,

      TemplateId = new Sitecore.Data.ID("{C3B8C43D-AD6B-49E2-9C6E-2FB4B53A966A}")
                        //sitecore/templates/MediaFramework/MyCompany/MyCompany Video
   }
   ;
```




## GetRootItem()

| | | |
|---|---|---|
| **Method:** | `public abstract Item GetRootItem(object entity, Item accountItem);` | |
| **Description:** | This method returns the Sitecore folder that contains all the media items. | |
| **Parameters** | | |
| **Name** | **Type** | **Description** |
| *entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2. |
| *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | |
| **Type:** | `Sitecore.Data.Items.Item` | |
| **Description:** | The Sitecore item that represents the parent folder for all video content. | |

The following example, shows how to retrieve the parent folder for the media content.

```
public override Item GetRootItem(object entity, Item accountItem)
{
   return accountItem.Children["Media Content"];
   // same as: accountItem.Children.FirstOrDefault(i => i.Name == "Media Content");
}
```

## GetSearchResult()

| | |
|---|---|
| **Method:** | `public abstract MediaServiceSearchResult GetSearchResult(object entity, Item accountItem);` |
| **Description:** | This method returns a Sitecore search result that matches the corresponding entity. |

| Parameters | | | |
|---|---|---|---|
| | **Name** | **Type** | **Description** |
| | *entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2. |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |
| | **Type:** | `Sitecore.MediaFramework.Entities.MediaServiceSearchResult` | |
| | **Description:** | The returned `MediaServiceSearchResult` is a Sitecore SearchResultItem that matches the corresponding entity item (i.e. Video).<br>For details on configuring/using Sitecore search see Search and Indexing Guide on http://SDN.sitecore.net. You may also find the following blogs helpful:<br>• http://www.sitecore.net/Community/Technical-Blogs/Sitecore-7-Development-Team/Posts/2013/06/Sitecore-7-POCO-Explained.aspx<br>• http://www.sitecore.net/Community/Technical-Blogs/Getting-to-Know-Sitecore/Posts/2013/06/Using-Luke-to-Understand-Sitecore-7-Search.aspx | |

In the following example, a data item is found by matching its template Id and unique Id field for the media element.

```
public override MediaServiceSearchResult GetSearchResult(object entity, Item accountItem)
{
   var videoPoco = entity as MyCompany.Entities.Video;

   var videoTemplateId  /* MyCompany Video template from section 2.1.5 */
         = new Sitecore.Data.ID("{C3B8C43D-AD6B-49E2-9C6E-2FB4B53A966A}");

   return base.GetSearchResult<MyCompany.Entities.VideoSearchResult>(
           indexName:   "mediaframework mycompany index"
         , accountItem: accountItem
         , selector:     i => (    (i.TemplateId   == videoTemplateId)
                         && (i.MyReadOnlyId == videoPoco.MyReadOnlyId))
         );

}
```

## NeedUpdate()

| **Method:** | `public abstract bool NeedUpdate(object entity, Item accountItem, MediaServiceSearchResult searchResult);` | |
|---|---|---|
| **Description:** | This method compares the media element retrieved from Import Executer (see section 3.2) to Sitecore item (i.e. search result). If there exists discrepancy between the two items, then true is returned; otherwise, false is returned. | |
| **Parameters** | | |
| **Name** | **Type** | **Description** |

| | | | |
|---|---|---|---|
| | *entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2. |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| | *searchResult* | `Sitecore.MediaFramework.Entities.MediaServiceSearchResult` | It is a Sitecore SearchResultItem that matches the corresponding entity item. For details on configuring/using Sitecore search see Search and Indexing Guide on http://SDN.sitecore.net. |
| **Return value** | | | |
| | **Type:** | `System.Boolean` | |
| | **Description:** | Returns true if any of the fields within the Sitecore item do not match the entity media content from service provider; otherwise, false is returned. | |

In the following example, the method compares record timestamp; in production environment it is beneficial to compare specific fields (as appose to updated timestamp) to insure that updates to non-imported fields do not trigger a synchronization process.

```
public override bool NeedUpdate(object entity, Data.Items.Item accountItem
      , Sitecore.MediaFramework.Entities.MediaServiceSearchResult searchResult)
{
   // determine if searched item is different than external resource
   var video = entity as MyCompany.Entities.Video;

   var videoSearchResult = searchResult as MyCompany.Entities.VideoSearchResult;

   return video.Updated > videoSearchResult.Updated;
}
```

## UpdateItem()

| | | | |
|---|---|---|---|
| **Method:** | `public abstract Item UpdateItem(object entity, Item accountItem, Item item);` | | |
| **Description:** | Just as `NeedUpdate()` method is responsible for determining if the synchronization is needed, the `UpdateItem()` is responsible for actual synchronization of an item from the service provider to Sitecore. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2. |

| | | | |
|---|---|---|---|
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| | *searchResult* | `Sitecore.MediaFramework.Entities.MediaServiceSearchResult` | It is a Sitecore SearchResultItem that matches the corresponding entity item. For details on configuring/using Sitecore search see Search and Indexing Guide on http://SDN.sitecore.net. |
| **Return value** | | | |
| | **Type:** | `Sitecore.Data.Items.Item` | |
| | **Description:** | Return the instance of Sitecore item that has been updated. | |

Below is a sample implementation of this method:

```
public override Data.Items.Item UpdateItem( object entity
                                          , Data.Items.Item accountItem
                                          , Data.Items.Item item)
{
   var video = entity as MyCompany.Entities.Video;

   using (new EditContext(item))
   {
      item.Fields[FieldIds.FileName].Value = video.FileName;
      item.Fields[FieldIds.Title].Value = video.Title;
      item.Fields[FieldIds.Description].Value = video.Description;
      // ...
   }

   return item;
}
```

## SyncItem()

| | |
|---|---|
| **Method:** | `public virtual Item SyncItem(object entity, Item accountItem)` |
| **Description:** | This method is the driver for updating Sitecore items; or create them if they do not already exist. This method "initiates" the MediaSyncItemImportPipeline for the specified media entity type that is responsible for synchronization. At the time of writing this document, the pipeline first issues a search to find a matching Sitecore item for the corresponding entity. If no item is found, then DatabaseFallback is used to find a match; lastly, CreateItem within the pipeline is called if the item is not found in the Sitecore database. For a full list of pipeline elements see Sitecore.MediaFramework.config file located in Configuration > Sitecore > mediaFramework > mediaFramework > mediaSyncItemImport.<br><br>**Note**<br>When Sitecore media item is updated, its corresponding export operations are queued until Sitecore.Media Framework.ExportTimeout (default 20 seconds) is reached. To avoid premature synchronization, check to see if there is a pending |

| | |
|---|---|
| | item in the queue for the export operation prior to calling the `base.SyncItem()` method. See below example. |

| **Parameters** | | | |
|---|---|---|---|
| | **Name** | **Type** | **Description** |
| | *entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2. |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |
| | **Type:** | `Sitecore.Data.Items.Item` | |
| | **Description:** | Sitecore item that corresponds to the media entity. | |

Below is sample implementation that overrides this method:

```
public override Item SyncItem(object entity, Item accountItem)
{
   var videoPoco = entity as MyCompany.Entities.Video;

   var myReadOnlyItemId = new Sitecore.Data.ID("{1B83DF17-DE85-49BB-A742-D199F0FDEA72}");

   if (ExportQueueManager.IsExist(accountItem, myReadOnlyItemId, videoPoco.MyReadOnlyId))
   {
      return null;
   }

   return base.SyncItem(entity, accountItem);
}
```

**Note**
It is recommended that developers override the `SyncItem()` method, similar to the above example, to inspect if an item is already queue for an export.

## CreateEntity()

| **Method:** | `public virtual object CreateEntity(Item item)` |
|---|---|
| **Description:** | Returns an instance of media element using the Sitecore item. For more information see section 5.2.2. |

| **Parameters** | | | |
|---|---|---|---|
| | **Name** | **Type** | **Description** |
| | *item* | `Sitecore.Data.Items.Item` | Sitecore item that needs to be converted into corresponding .NET media entity object. |
| **Return value** | | | |
| | **Type:** | `System.Objects` | |

| | | |
|---|---|---|
| | **Description:** | A media item object instance that corresponds to the Sitecore item. This object is one of the .NET classes created for exporters and importers in section 3.2.2 and 4.2.2. |

## Fallback()

| | | | |
|---|---|---|---|
| | **Method:** | `public virtual MediaServiceSearchResult Fallback(object entity, Item accountItem)` | |
| | **Description:** | Returns Sitecore search result that matches the media element. This is used when Sitecore index search returns nothing, which can occur if the underlying search engine (i.e. Lucene, Solar, Coveo, etc) index is not up-to-date. For more information see section 5.2.3. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2 |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |
| | **Type:** | `Sitecore.MediaFramework.Entities.MediaServiceSearchResult` | |
| | **Description:** | The returned `MediaServiceSearchResult` is a Sitecore SearchResultItem that matches the corresponding entity item.<br>For details on configuring/using Sitecore search see Search and Indexing Guide on http://SDN.sitecore.net. You may also find the following blogs helpful:<br>• http://www.sitecore.net/Community/Technical-Blogs/Sitecore-7-Development-Team/Posts/2013/06/Sitecore-7-POCO-Explained.aspx<br>• http://www.sitecore.net/Community/Technical-Blogs/Getting-to-Know-Sitecore/Posts/2013/06/Using-Luke-to-Understand-Sitecore-7-Search.aspx | |

## AddReference()

| | | | |
|---|---|---|---|
| | **Method:** | `public void AddReference(XmlNode node)` | |
| | **Description:** | Updates the References property of the abstract class to satisfy `IItemSynchronizer` by creating an instance of synchronizer references using Sitecore factory class based on the configuration file (see section 5.2.5). For more information see Implement Field Reference in see section 5.2.4. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *node* | `System.Xml.XmlNode` | Xml dom object that points to custom configuration file. |
| **Return value** | | | |
| | **Type:** | `Void` | |

| | | |
|---|---|---|
| | **Description:** | N/A |

## 5.2.2 Implement Entity Creator

Entity creators are responsible for translating a Sitecore item into a media entity. The interface `Sitecore.MediaFramework.Synchronize.IMediaServiceEntityCreator` has a single method that must be implemented.

### CreateEntity()

| | | | |
|---|---|---|---|
| **Method:** | `public object CreateEntity(Item item)` | | |
| **Description:** | Converts a Sitecore item to .NET class that represents the external service provider media element. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *item* | `Sitecore.Data.Items.Item` | Sitecore item that needs to be converted into corresponding .NET media entity object. |
| **Return value** | | | |
| | **Type:** | `System.Object` | |
| | **Description:** | A media item object instance that corresponds to the Sitecore item. This object is one of the .NET classes created for exporters and importers in section 3.2.2 and 4.2.2 | |

Below example demonstrates the implementation for this method. Note that the `FieldIds` is a static class that contains Sitecore IDs for all the template fields.

```
public class PlayerEntityCreator : IMediaServiceEntityCreator
{
   public object CreateEntity(Item item)
   {
      var player = new MyCompany.Entities.Player()
      {
      Id = item.Fields[FieldIds.PlayerId].Value,
      Name = item.Fields[FieldIds.PlayerId].Value,
      BorderColor = item.Fields[FieldIds.Title].Value,
      AutoRestart = item.Fields[FieldIds.Description].Value,
      };
   return player;
}
```

## 5.2.3 Implement DatabaseFallbackBase

Database fallback is used when the Sitecore search API (using an indexed search) yields to no result set. This can happen if the content index is not up-to-date. Developers must implement two methods to satisfy the abstract class `Sitecore.MediaFramework.Synchronize.Fallback.DatabaseFallbackBase`.

The following section describes the `DatabaseFallbackBase` methods:

## GetItem()

| | | | |
|---|---|---|---|
| **Method:** | `protected override Item GetItem(object entity, Item accountItem)` | | |
| **Description:** | Finds a Sitecore item that corresponds to the media entity passed in. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2 |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |
| | **Type:** | `System.Object` | |
| | **Description:** | A Sitecore media item that corresponds to entity passed in. This object is one of the .NET classes created for exporters and importers in section 3.2.2 and 4.2.2 | |

Below example demonstrates how to find and return a matching Sitecore item without an index search:

```
protected override Item GetItem(object entity, Item accountItem)
  {
      var videoTemplateId = new ID("{C3B8C43D-AD6B-49E2-9C6E-2FB4B53A966A}");

      var videoPoco = entity as MyCompany.Entities.Video;

      var queryString = string.Format(
            "./Media Content/*[@@templateid='{0}' and @MyReadonlyId='{1}']"
            , videoTemplateId
            , videoPoco.MyReadOnlyId);

      return accountItem.Axes.SelectSingleItem(queryString);
  }
```

## GetSearchResult()

| | | | |
|---|---|---|---|
| **Method:** | `protected override MediaServiceSearchResult GetSearchResult(Item item)` | | |
| **Description:** | Converts a Sitecore item to .NET class that represents the external service provider media element. The abstract class uses this method to convert the item retrieved from `GetItem()` method. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *item* | `Sitecore.Data.Items.Item` | Sitecore item that needs to be converted into `MediaServiceSearchResult`. |
| **Return value** | | | |
| | **Type:** | `Sitecore.MediaFramework.Entities.MediaServiceSearchResult` | |
| | **Description:** | Returns an instance of `MediaServiceSearchResult` where its values are set from the parameter item. | |

Below example demonstrates the implementation for this method:

```
protected override MediaServiceSearchResult GetSearchResult(Item item)
{
    var searchResult =
            Activator.CreateInstance<MyCompany.Entities.PlayerSearchResult>();

    searchResult.Id = item.Fields[FieldIds.PlayerId].Value;

    searchResult.Name = item.Fields[FieldIds.PlayerName].Value;

    // ...

    return searchResult;
}
```

## FillStandardProperties()

| | | | |
|---|---|---|---|
| **Method:** | `protected virtual void FillStandardProperties(MediaServiceSearchResult searchResult, Item item)` | | |
| **Description:** | Populates searchResult parameter with additional item properties. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *searchResult* | `Sitecore.MediaFramework.Entities.MediaServiceSearchResult` | Reference parameter that will have its base fields updated (i.e. Name, DatabaseName, Language, Version, Uri). |
| | *item* | `Sitecore.Data.Items.Item` | Sitecore item that is used as a source to update the search result instance. |
| **Return value** | | | |
| | **Type:** | `Void` | |
| | **Description:** | N/A | |

Below example demonstrates the implementation for this method:

```
protected override MediaServiceSearchResult GetSearchResult(Item item)
{
    var searchResult =
            Activator.CreateInstance<MyCompany.Entities.PlayerSearchResult>();

    searchResult.Id = item.Fields[FieldIds.PlayerId].Value;

    searchResult.Name = item.Fields[FieldIds.PlayerName].Value;

    // ...

    return searchResult;
}
```

### 5.2.4 Implement Field Reference IdReferenceSynchronizer

For any given synchronizer, you may specify zero or more field reference synchronizers. Reference synchronizers are used to satisfy cascade update to dependent elements. For example, when video media elements change, then players with playlist of video items need to be updated. Typically, reference synchronizers are used in the following cases:

- To synchronize item references in list field types, such as: like Multi-list, Tree-list and etc.
- To fill item fields by issuing additional requests to the external media service provider.

Reference synchronizers inherit from the abstract class `Sitecore.MediaFramework.Synchronize.References.IdReferenceSynchronizer<Entity>` and provide the implementation definition for `GetReference()` method.
The abstract class `IdReferenceSynchronizer` provides the implementation details for `IReferenceSynchronizer.SyncReference()`, which it first finds the specific reference field by calling the abstract method `GetField()`; then it retrieves the object `List<ID>` using the abstract method `GetReference();` and finally it calls `UpdateField()` when the respective `NeedUpdate()` method returns true. The synchronizer method `GetField()` returns the field ID that is specified in the configuration file (i.e. see player node in the configuration file in section 5.2.5).

#### GetReference()

| | | | |
|---|---|---|---|
| **Method:** | `protected override List<ID> GetReference(Entity entity, Item accountItem)` | | |
| **Description:** | Returns a list of Sitecore IDs that correspond to matching external media entity. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *entity* | `System.Object` | Entity corresponds to respective .NET class object created for exporters and importers in section 3.2.2 and 4.2.2. |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |
| | **Type:** | `List<Sitecore.Data.ID>` | |
| | **Description:** | If a Sitecore item is found that matches the respective entity, then its ID is returned in a list containing a single item. Otherwise, an empty list is returned. | |

Note in the following example, Ooyala uses the class `Asset` as a base class for all its media types.

```
public class AssetPlayerSynchronizer : IdReferenceSynchronizer<Asset>
{
  protected override List<ID> GetReference(Asset entity, Item accountItem)
  {
    var parameters = new List<Parameter>
                     {
                        new Parameter()
                        {
                           Name = "embedcode",
                           Type = ParameterType.UrlSegment,
                           Value = entity.EmbedCode
```

```
                            }
                        };

            var context = new RestContext(Constants.SitecoreRestSharpService
                                        , new OoyalaAthenticator(accountItem));

            var referencedPlayer = context.Read<ReferencedPlayer>("read_asset_player"
                                                                , parameters).Data;
            if (referencedPlayer == null)
            {
               return new List<ID>(0);
            }

            var playerIndex = ContentSearchUtil.FindOne<PlayerSearchResult>(Constants.IndexName
                            , i =>    i.Paths.Contains(accountItem.ID)
                                   && i.TemplateId == TemplateIDs.Player
                                   && i.Id == referencedPlayer.ID
                            );

            // Use fallback search when index search yields to nothing
            //
            if (playerIndex == null)
            {
               IItemSynchronizer synchronizer
                  = MediaFrameworkContext.GetItemSynchronizer(typeof(Player));

               if (synchronizer != null)
               {
                  Player player = new Player {Id = referencedPlayer.ID};

                  playerIndex = synchronizer.Fallback(player, accountItem) as PlayerSearchResult;
               }
            }
            return playerIndex != null ? new List<ID> { playerIndex.ItemId } : new List<ID>(0);
         }
      }
```

### 5.2.5 Define Synchronizer

The base exporter reads synchronization executers from the Sitecore configuration files. Synchronizers must be defined under the `configuration > sitecore > mediaFramework > synchronizers` section.

Each synchronizer is added using four attributes, which include: `name`, `entity`, `templateId`, and `type`. The `name` attribute is the string name that uniquely identifies the synchronizer among other synchronizers. The `templateId` attribute is the media item templateId created in section 2.1.5. The `entity` attribute is the fully-qualified type name for the class that corresponds to the Sitecore media item. Lastly, the `type` attribute is the fully-qualified type name for the class that implements the entity synchronizer.

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

Below configuration demonstrates Ooyala synchronizer, which is invoked when corresponding video exporter is initiated. Upon completion of the export, the synchronizer checks other Sitecore item fields (such as Labels, player list, metadata) to update respective elements by comparing field values to that of the service provider.

```
      <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
         <sitecore>
            <mediaFramework>
```

```
                <synchronizers>
                   <!-- For example, partial video synchronizer for Ooyala service.

                   <add name="ooyala video"
                        entity="Sitecore.MediaFramework.Ooyala.Entities.Video,
Sitecore.MediaFramework.Ooyala"
                        templateId="{C3B8C43D-AD6B-49E2-9C6E-2FB4B53A966A}"
                        type="Sitecore.MediaFramework.Ooyala.Synchronize.VideoSynchronizer,
Sitecore.MediaFramework.Ooyala">

                      <entityCreator
type="Sitecore.MediaFramework.Ooyala.Synchronize.EntityCreators.VideoEntityCreator,
Sitecore.MediaFramework.Ooyala"/>

                      <databaseFallback
type="Sitecore.MediaFramework.Ooyala.Synchronize.Fallback.VideoFallback,
Sitecore.MediaFramework.Ooyala"/>

                      <references hint="raw:AddReference">
                         <player
type="Sitecore.MediaFramework.Ooyala.Synchronize.References.AssetPlayerSynchronizer,
Sitecore.MediaFramework.Ooyala">
                            <field>{6619DFDB-A4A7-41FA-86EA-42CFBB6F25C4}</field>
                         </player>
                      </references>
                   </add>
                   -->
                </synchronizers>
             </mediaFramework>
          </sitecore>
       </configuration
```

In the above synchronizer configuration four xml nodes are defined, which include:

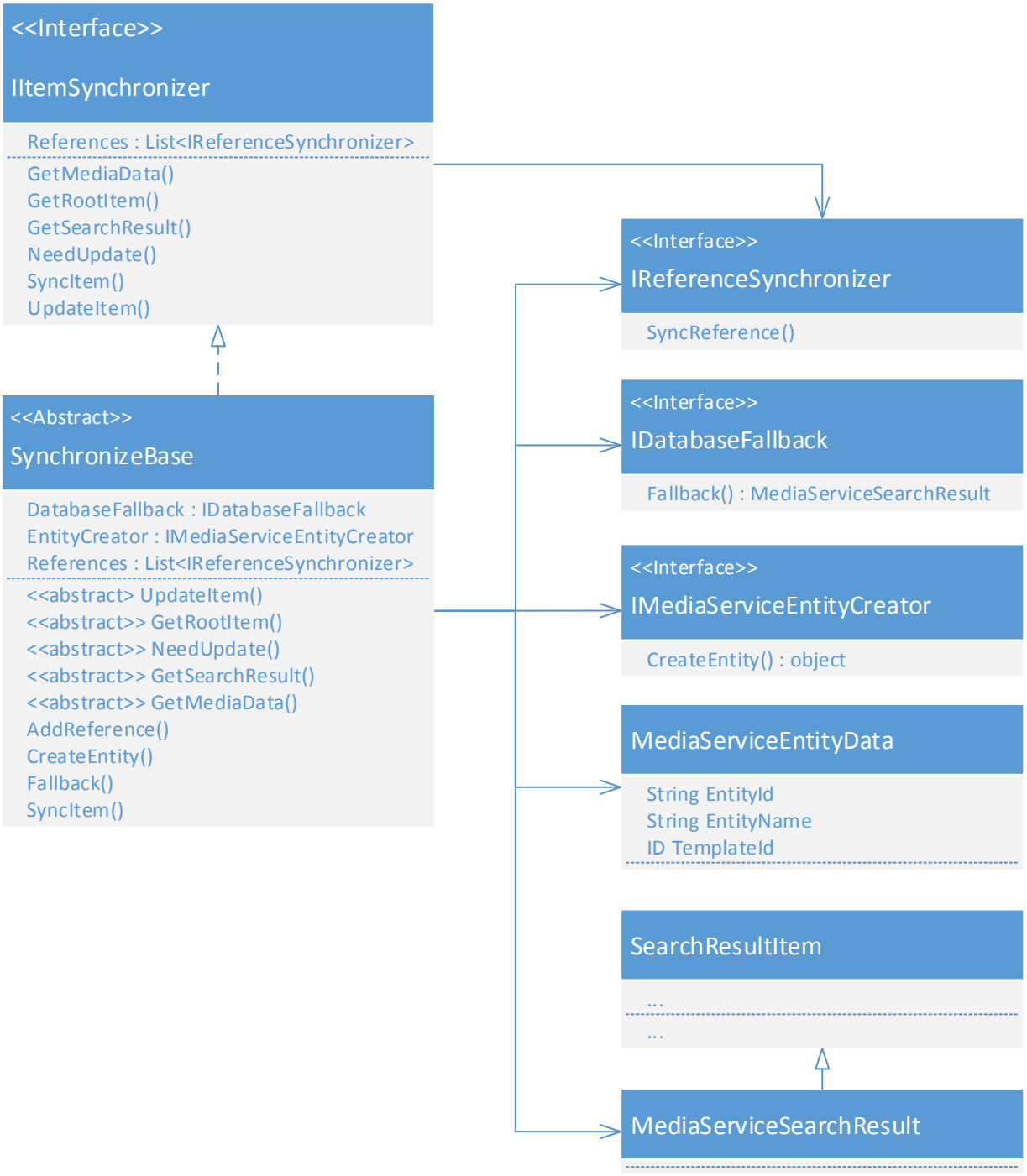
- The `add` node is responsible for specifying the synchronizer implementation. As mentioned above, the `add` node has four attributes `name`, `templateId`, `entity`, and `type`. The `name` attribute is the string name that uniquely identifies the synchronizer among other synchronizers. The `templateId` attribute is the media item templateId created in section 2.1.5. The `type` attribute is the fully-qualified type name for the class that implements the synchronizer. Lastly, the `entity` attribute is the fully-qualified .NET entity class that was specified for importer or exporter in section 3.2.2and 4.2.2 respectively.
- The `entityCreator` node has one `type` attribute, which is the fully-qualified type name for the class that implements the entity creation (see section 5.2.2).
- The `databaseFallback` node has one `type` attribute, which is the fully-qualified type name for the class that implements the respective class (see section5.2.3).
- The `reference` node has zero or more references to other field references. Field references are used to check the dependency between exported items to external service provider and other Sitecore items that represent different media elements.
  The references child node name is unimportant; however, it must contain the full qualified path to the respective reference (see section 5.2.4). Each child node has a `field` reference which points to the field ID of the respective template item. The `field` value will hydrate `FieldReferenceSynchronizer<TEntity,TRresult>`, which is the root base class to the field reference. Sitecore will map the node name to the respective property of the field referencer. In above example, the "`ooyala_video`" synchronizer references player field within the media data item to update list of players for a Droplink field type.

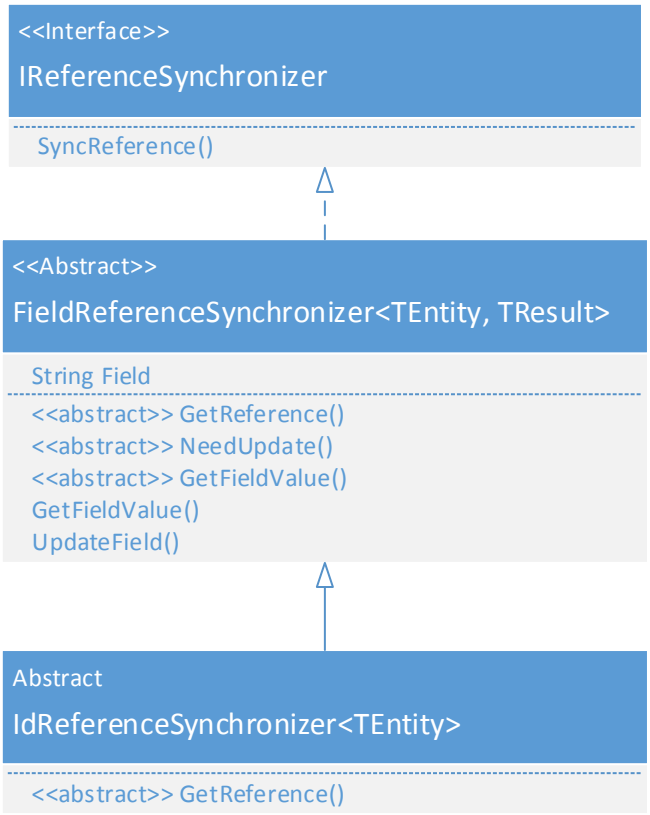
  To specify more than one field for an item, without further customization, the referencer must specify the media item configuration element multiple times. For example, if there was a need

to update more than one field for a given player, the configuration from above would be revised as following:

```
...
                <references hint="raw:AddReference">

                   <player type="[Full Assembly Path]">
                      <field>{GUID1}</field>
                   </player>

                   <player type="[Full Assembly Path]">
                      <field>{GUID2}</field>
                   </player>

                   <player type="[Full Assembly Path]">
                      <field>{GUID2}</field>
                   </player>

                </references>
...
```

### 5.2.6 Class Diagram





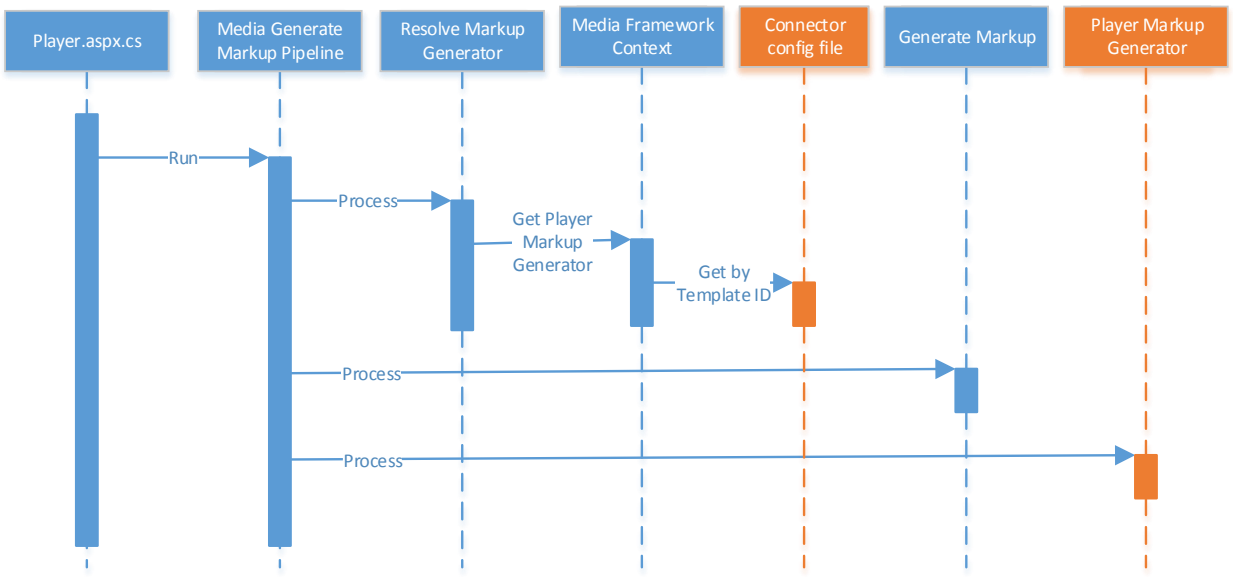
# Chapter 6

# Media Player Markup

This chapter covers the HTML markup and JavaScript implementation for a Media Framework connector.

## 6.1 Overview

The following sequence diagram illustrates when the generator is used in a webpage rendering. The orange boxes represent the components that must be implemented. The blue boxes are part of Media Framework, and do not need to be changed.



## 6.2 Markup Generators

Markup generators provide implementation details necessary to render a video player on a webpage for the external media service provider.

### 6.2.1 Implement PlayerMarkupGeneratorBase

Developer(s) must create at least one video player by inheriting from the abstract class `Sitecore.MediaFramework.Players.PlayerMarkupGeneratorBase`, which implements the interface `Sitecore.MediaFramework.Players.IPlayerMarkupGenerator`. The abstract class has the following methods:

#### GetMediaId()

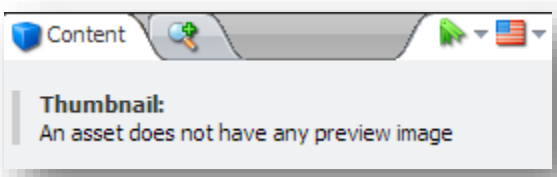
| | | | |
|---|---|---|---|
| **Method:** | `public abstract string GetMediaId(Item item);` | | |
| **Description:** | Gets the unique media identifier specified by the external media service provider for the specified item. Recall, that this is a read-only item attribute that is set during import synchronization of an entity. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *item* | `Sitecore.Data.Items.Item` | The Sitecore item that contains the unique media Id recognized by the external service provider. |
| **Return value** | | | |
| | **Type:** | `System.String` | |
| | **Description:** | Unique media id that is recognized by external media provider. | |

In the following example, `MyReadOnlyId` variable is the service provider's unique identifier for the respective media item. The static class `FieldIds` is a helper class that contains reference to Sitecore field IDs.

```
public override string GetMediaId(Item item)
{
    return item.Fields[FieldIds.MyReadOnlyId].Value;
}
```

#### GetPreviewImage()

| | |
|---|---|
| **Method:** | `public abstract string GetPreviewImage(MediaGenerateMarkupArgs args);` |
| **Description:** | Returns html `IMG` markup to provide preview of media instead of the player (required in some situations).<br>Note that content Editor shows an image of the medial item via `GetPreviewImage()` method; otherwise, it displays a message that the asset does not have a preview image. |

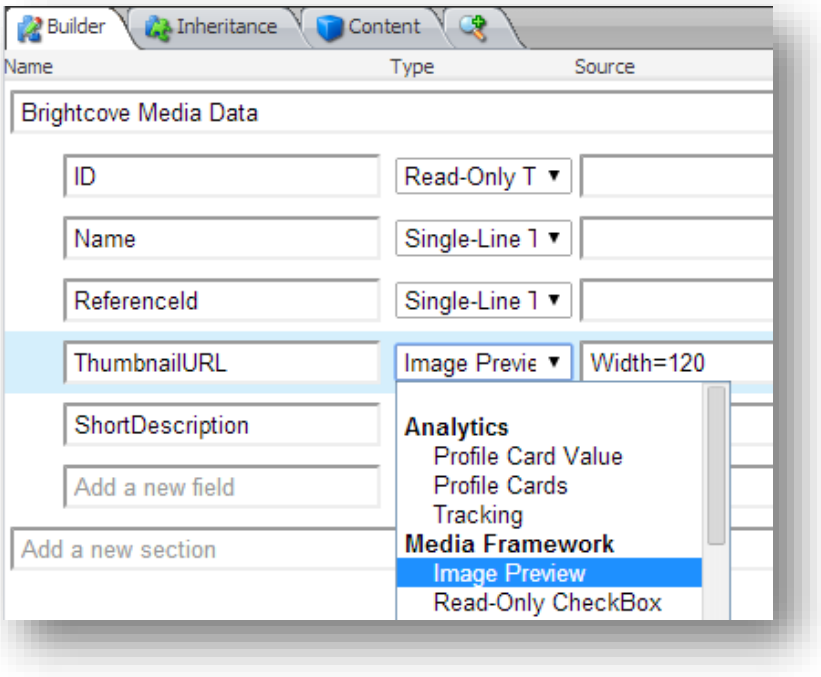



| Parameters | | | |
|---|---|---|---|
| | **Name** | **Type** | **Description** |
| | *Args* | `Sitecore.MediaFramework.Pipelines.MediaGenerateMarkup.MediaGenerateMarkupArgs` | `MediaGenerateMarkupArgs` class is described later. |
| **Return value** | | | |
| | **Type:** | `System.String` | |
| | **Description:** | Html markup to show a video element as an image for a preview. | |

The following code snippet uses `PlayerManager` provider to return image markup that points to a Sitecore field, which is defined as "Image Preview" field type.

```
public override string GetPreviewImage(MediaGenerateMarkupArgs args)
{
    return PlayerManager.GetPreviewImage(args, FieldIDs.MediaElement.ThumbnailUrl);
}
```

Note, that "Image Preview" fields are string values that represents full URL to a thumbnail image. This field type can be updated programmatically via Item Synchronizer `UpdateItem()` method.



The parameter type `MediaGeneratedMarkupArgs` (from above) is inherited from `PipelineArgs`. Below we list the properties that are defined in this `MediaGeneratedMarkupArgs`.

- `public MarkupType MarkupType { get; set; }`
  Enumerated type value that describes the generator markup type, where `MarkupType` is defined as: `enum MarkupType {Frame, FrameUrl, Link, Html }`

- `public IPlayerMarkupGenerator Generator { get; set; }`
  This property corresponds to the Markup Generators created earlier in section 6.2.

- `public PlayerProperties Properties { get; set; }`
  The player property definition is shown below:

```
public class PlayerProperties
{
    public ID Template {get; set;}

    public ID ItemId {get; set;}

    public ID PlayerId {get; set;}

    public string MediaId{get; set;}

    public int Width {get; set;}

    public int Height {get; set;}

    public bool ForceRender{get; set;}
}
```

- `public Database Database { get; set; }`
  Sitecore Context Database for the item.

- `public Item MediaItem { get; set; }`
  Sitecore media Item to be used for the player.

- `public Item AccountItem { get; set; }`
  The parameter `accountItem` is the Sitecore object that references the Account element created by user.

- `public Item PlayerItem { get; set; }`
  Siteccore player Item associated with this markup generator.

- `public string LinkTitle { get; set; }`
  Returns text “Media Link” when `this.MarkupType` value is `Link`; otherwise, it is null.

- `public Dictionary<string, List<string>> PlaybackEvents { get; set; }`
  Contains a list of playback events where the dictionary key corresponds to events like: PlaybackStarted, PlaybackCompleted, etc. Currently, the value for each key is a list containing an empty string. Media Event Triggers is discussed in Chapter 7.

- `public PlayerMarkupResult Result { get; set; }`
  See `Generate()` method.

## GetDefaultPlayer()

| | | | |
|---|---|---|---|
| **Method:** | `public abstract Item GetDefaultPlayer(MediaGenerateMarkupArgs args);` | | |
| **Description:** | Returns Sitecore player item that corresponds to the video. Typically, there are only a few (one or two) players for all the media sources. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `Sitecore.MediaFramework.Pipelines.MediaGenerateMarkup.MediaGenerateMarkupArgs` | See `GetPreviewImage()` method argument. |
| **Return value** | | | |
| | **Type:** | `Sitecore.Data.Items.Item` | |
| | **Description:** | Sitecore item that is used to hold media player information. | |

Ooyala has only one player and uses the following expression to return the default player:

```
public override Item GetDefaultPlayer(MediaGenerateMarkupArgs args)
{
    return AccountManager.GetPlayers(args.AccountItem)
        .FirstOrDefault(player => player[FieldIDs.Player.IsDefault] == "1");
}
```

In the above expression, the index reference `FieldIDs.Player.IsDefault` is a GUID for a specific field within a player template.

Brightcove on the other hand offers two players and uses the following code snippet to differentiate between video player and playlist player. Notice that Brightcove uses field Ids that are set under Create Default Account Settings Item (section 2.2.1)

```
public override Item GetDefaultPlayer(MediaGenerateMarkupArgs args)
{
    ID fieldId = args.MediaItem.TemplateID == TemplateIDs.Video
                    ? FieldIDs.AccountSettings.DefaultVideoPlayer
                    : FieldIDs.AccountSettings.DefaultPlaylistPlayer;

    var settingsField = AccountManager.GetSettingsField(args.AccountItem, fieldId);

    return settingsField != null ? settingsField.TargetItem : null;
}
```

## Generate()

| | | | |
|---|---|---|---|
| **Method:** | `public abstract PlayerMarkupResult Generate(MediaGenerateMarkupArgs args);` | | |
| **Description:** | Returns a class that contains the required HTML, CSS, and JavaScript needed to render a video element. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `Sitecore.MediaFramework.Pipelines.MediaGenerateMarkup.MediaGenerateMarkupArgs` | See `GetPreviewImage()` method argument. |
| **Return value** | | | |
| | **Type:** | `Sitecore.MediaFramework.Pipelines.MediaGenerateMarkup.PlayerMarkupResult` | |

**Description:**

Returning an instance of `PlayerMarkupResult`, which contains the following properties:

- `public Dictionary<string, string> BottomScripts { get; private set; }`
  This is a collection of JavaScript text that is added to the end of player markup. The collection name is used as a unique item identifier. For example, you can use the following expression to add a JavaScript that is unique to each player within a webpage:

```
ScriptUrls.Add("~/JS/<ServiceProvider>.js");

var scriptName = string.Format("<ServiceProvider>_{0}"
                               , args.MediaItem.ID);
var javascript = "/*...*/";

myPlayerMarkupResult.BottomScripts.Add(scriptName, javascript);
```

  Media Framework uses this property to add client side events associated with respective player item.

- `public string Html { get; set; }`
  This is the HTML markup used to render the video player markup.

- `public List<string> CssUrls { get; private set; }`
  This is a list of CSS URLs to include before the player markup, which is specified using `Html` property.

- `public List<string> ScriptUrls { get; private set; }`
  This is a list of JavaScript URLs to include before the player , which is specified using `Html` property.

**Note**
`PlayerMarkupResult` is utilized to render a sublayout; however, some items like CSS and JavaScripts are added to the html `<HEADER>` section of a webpage. Additionally, when the same resource item appears more than once, only one instance of the item is added to the header (i.e. using multiple renderings in a page). Resources are added using `PlayerManager` provider, which is defined in Sitecore.MediaFramework.config file.

**Note**
Instead of hardcoding resource URLs, consider retrieving them from the configuration file. You can reference items from the configuration file by simply creating public properties and then updating the configuration file with the matching names. For example the property `public ScriptUrl {get; set;}` will be populated via the following configuration:

```
<playerMarkupGenerators>
   <add name="ooyala_video" ...>
    <scriptUrl>http://player.ooyala.com...</scriptUrl>
    ...
```

## GenerateFrameUrl()

| | | | |
|---|---|---|---|
| **Method:** | `public virtual string GenerateFrameUrl(MediaGenerateMarkupArgs args)` | | |
| **Description:** | Returns the URL that is utilized for the `SRC` attribute of a html `IFRAME`. The default implementation points to a Sitecore sublayout using constant `Sitecore.MediaFramework.Constants.PlayerIframeUrl`, which contains the following string: `"/layouts/MediaFramework/Sublayouts/Player.aspx"` This function is utilized in the `GetFrame()` method. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `Sitecore.MediaFramework.Pipelines.MediaGenerateMarkup.MediaGenerateMarkupArgs` | See `GetPreviewImage()` method argument. |
| **Return value** | | | |
| | **Type:** | `System.String` | |
| | **Description:** | Returns the URL that is utilized for the `SRC` attribute of a html `IFRAME` to show the video. | |

## GetFrame()

| | | | |
|---|---|---|---|
| **Method:** | `public virtual string GetFrame(MediaGenerateMarkupArgs args)` | | |
| **Description:** | Returns a HTML `IFRAME` markup that uses `GenerateFrameUrl()` method to contain the video. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `Sitecore.MediaFramework.Pipelines.MediaGenerateMarkup.MediaGenerateMarkupArgs` | See `GetPreviewImage()` method argument. |
| **Return value** | | | |
| | **Type:** | `System.String` | |
| | **Description:** | `IFRAME` html markup that points to the video source returned by `GenerateFrameUrl()` method | |

Below example demonstrates the implementation for this method

```
        public virtual string GetFrame(MediaGenerateMarkupArgs args)
        {
            return
            string.Format( @"<iframe scrolling='no' class='player-frame' width='{0}' height='{1}'
frameborder='0' src='{2}'></iframe>",
                args.Properties.Width,
                args.Properties.Height,
                this.GenerateFrameUrl(args));
        }
```

## GenerateLinkHtml()

| | |
|---|---|
| **Method:** | `public string GenerateLinkHtml(MediaGenerateMarkupArgs args)` |
| **Description:** | Returns HTML markup anchor tag that provides a hyperlink to a player. This function is called from `Sitecore.MediaFramework.Pipelines.MediaGenerateMarkup.Genera` |

| | | | |
|---|---|---|---|
| | | teMarkup pipeline, which is specified under mediaFramework.mediaGenerateMarkup within SitecoreMediaFramework.config file. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | Sitecore.MediaFramework.Pipelines.MediaGenerateMarkup.MediaGenerateMarkupArgs | See GetPreviewImage() method argument. |
| **Return value** | | | |
| | **Type:** | System.String | |
| | **Description:** | Anchor hyperlink tag to the IFRAME that is responsible to render the video. | |

The following code snippet shows the underlying implementation for this method:

```
public string GenerateLinkHtml(MediaGenerateMarkupArgs args)
{
    return string.Format(
        "<a class='mf_mediaLink iframe' href='{0}' frwidth='{2}' frheight='{3}' >{1}</a>"
        , this.GenerateFrameUrl(args)
        , args.LinkTitle
        , args.Properties.Width
        , args.Properties.Height
        );
}
```

## 6.2.2 Define Markup Generator

As with other executers, markup generators are read from the Sitecore configuration files. The Player markup generators must be defined under the configuration > sitecore > mediaFramework > playerMarkupGenrators section.

Generators are added via add node with three attributes: name, templateId, type. The name attribute is a unique identifier among other markup generators. The templateId attribute is the media item templateId, which was created in section 2.1.5. Lastly, the type attribute is the fully-qualified type name for the class that implements the markup generator.

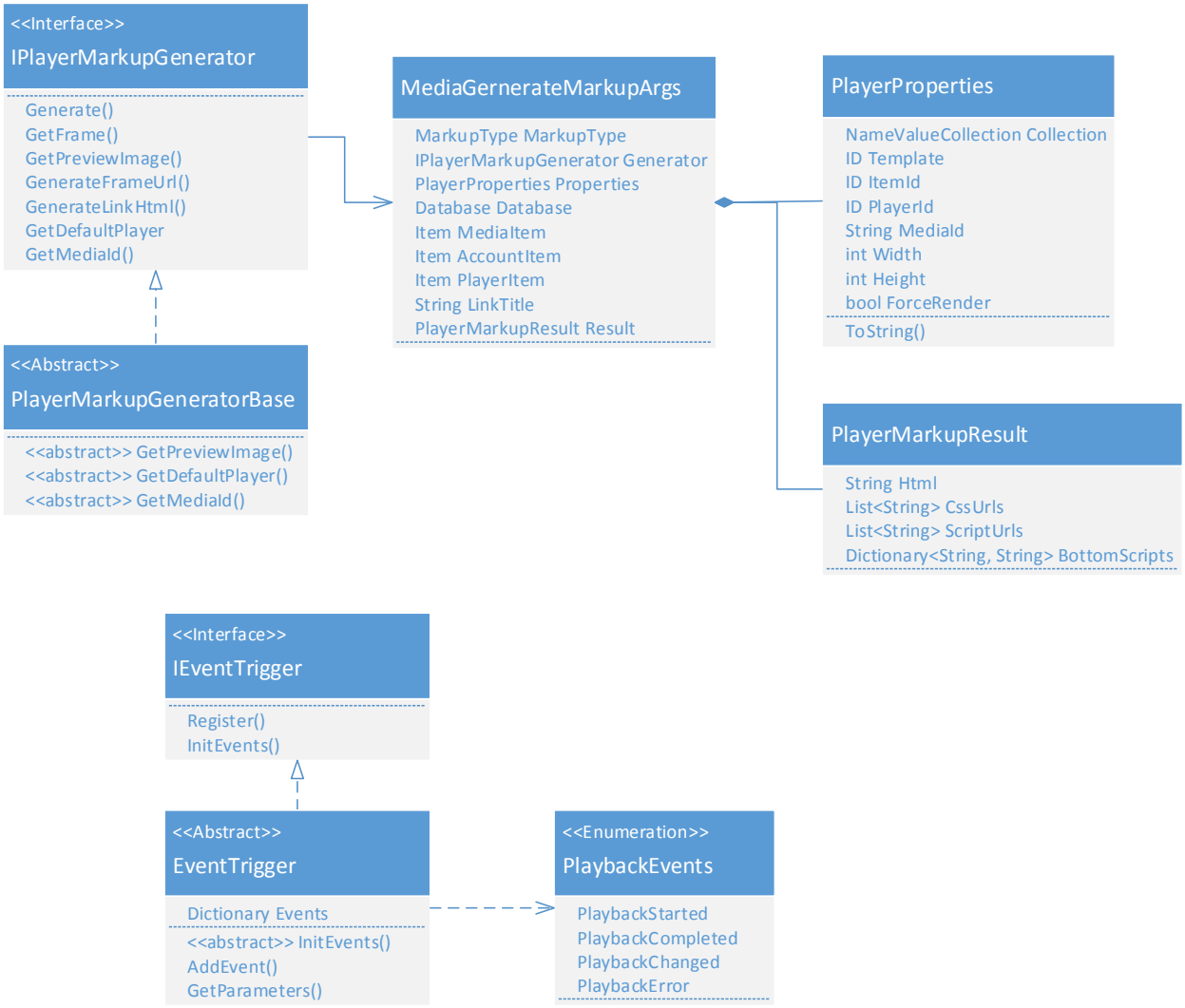
Additionally, you may set generator properties via configuration file by adding the property names as child nodes of the add node. For example, below configuration shows how Ooyala sets the property OoyalaPlayerMarkupGenerator.ScriptUrl property via scriptUrl xml node.

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <mediaFramework>

      <!-- ... -->

      <playerMarkupGenerators>
         <!-- For example, the player markup generator for Ooyala service. -->
         <add name="ooyala_video"
              templateId="{C3B8C43D-AD6B-49E2-9C6E-2FB4B53A966A}"

type="Sitecore.MediaFramework.Ooyala.Players.OoyalaPlayerMarkupGenerator,Sitecore.MediaFramework.
Ooyala">

                <scriptUrl>http://player.ooyala.com/v3/{0}?platform=html5-
fallback%26namespace={1}</scriptUrl>
          </add>
```

```
                <add name="<ServiceProvider>_<EntityName>"
                     templateId="<entity template ID>"

type="Sitecore.MediaFramework.<ServiceProvider>.Players.<ServiceProvider>PlayerMarkupGenerator,Si
tecore.MediaFramework.<ServiceProvider>">

                  <scriptUrl>[player script URL]</scriptUrl>
                </add>

              </playerMarkupGenerators>

              <!-- ... -->

            </mediaFramework>
          </sitecore>
        </configuration>
```

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

## 6.3 Class Diagram



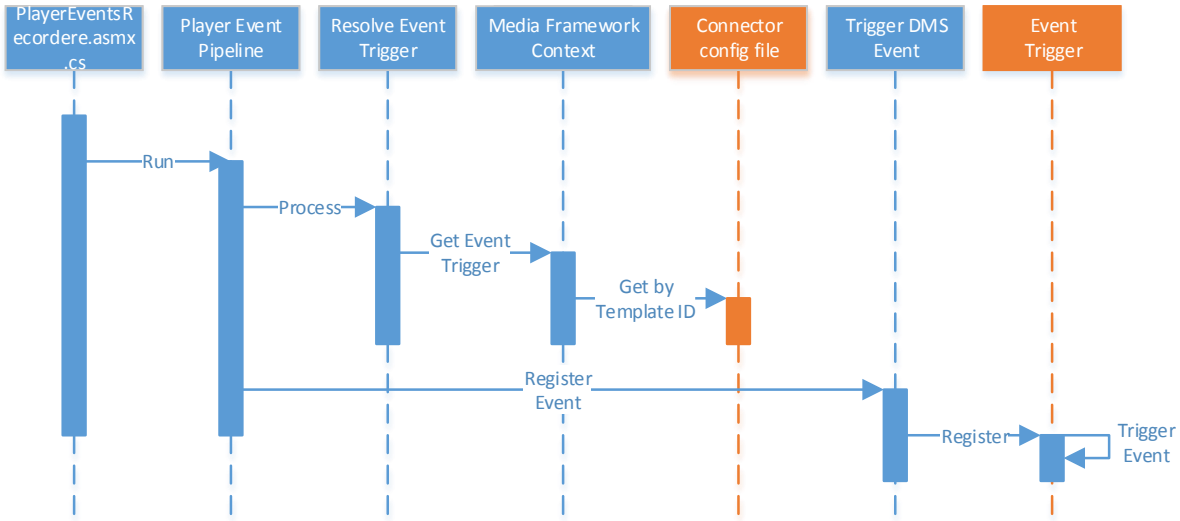
# Chapter 7

# Media Event Triggers

This chapter covers implementation requirement to leverage Customer Engagement Platform (CEP) for testing, personalization, and goal settings.

## 7.1 Overview

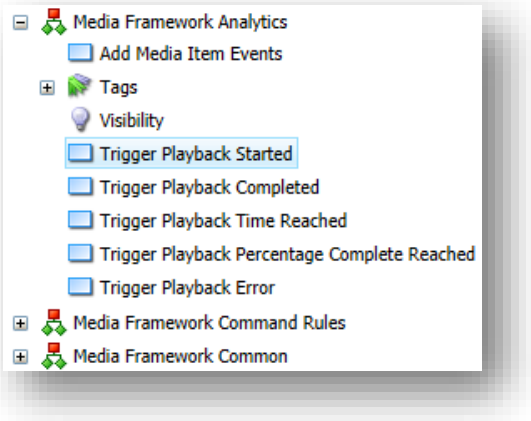
The following sequence diagram illustrates when the event triggers are used in Sitecore. The orange boxes represent the components that must be implemented. The blue boxes are part of Media Framework, and do not need to be changed.



## 7.2 Event Trigger

Server side event triggers are responsible for capturing events into the Sitecore Digital Marketing System (DMS). In Sitecore master database, event triggers are defined in the following path: `/sitecore/system/Settings/Rules/Definitions/Elements/Media Framework Analytics`. In this path you'll find the following events:

- Add Media Item Event
- Trigger Playback Started
- Trigger Playback Completed
- Trigger Playback Time Reached
- Trigger Playback Percentage Complete Reached
- Trigger Playback Error




### 7.2.1 Implement Server Side EventTrigger

Developer(s) must create media event triggers that inherit from the abstract class `Sitecore.MediaFramework.Analytics.EventTrigger`. The abstract class has the following methods:

#### InitEvents

| | | | |
|---|---|---|---|
| **Method:** | `public abstract void InitEvents();` | | |
| **Description:** | This methods registers list of events that should be supported. The registration is accomplished by calling the protected `AddEvent()` method. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `Sitecore.MediaFramework.Pipelines.Analytics.PlayerEventArgs` | Contains event specific information (i.e. Name, Properties) |
| **Return value** | | | |
| | **Type:** | `Void` | |
| | **Description:** | N/A | |

Following code snippet, shows Ooyala's implementation for registering supported events:

```
public class VideoEventTrigger : EventTrigger
{
    public override void InitEvents()
    {
      var videoTemplateId = new ID("{C3B8C43D-AD6B-49E2-9C6E-2FB4B53A966A}");

      this.AddEvent( videoTemplateId
                   , PlaybackEvents.PlaybackStarted.ToString()
                   , "Ooyala video is started.");

      this.AddEvent( videoTemplateId
                   , PlaybackEvents.PlaybackCompleted.ToString()
                   , "Ooyala video is completed.");
```

```
            this.AddEvent( videoTemplateId
                         , PlaybackEvents.PlaybackChanged.ToString()
                         , "Ooyala video progress is changed.");

            this.AddEvent( videoTemplateId
                         , PlaybackEvents.PlaybackError.ToString()
                         , "Ooyala video playback error.");
        }
    }
```

### AddEvent

| | | | |
|---|---|---|---|
| **Method:** | `protected virtual void AddEvent(ID templateId, string eventName, string text)` | | |
| **Description:** | Registers supporting DMS events for the playback. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *templateId* | `Sitecore.Data.ID` | Sitecore template ID that corresponds to the media item. |
| | *eventName* | `System.String` | The event name that is being registered. A list of enumerated event names are specified in `Sitecore.MediaFramework.PlaybackEvents`, which include: `PlaybackStarted`, `PlaybackCompleted`, `PlaybackChanged`, `PlaybackError` |
| | *text* | `System.String` | Human readable representation of the media event |
| **Return value** | | | |
| | **Type:** | `Void` | |
| | **Description:** | N/A | |

## 7.2.2 Define Event Trigger

The Event Triggers are defined in Sitecore configuration files under the `configuration > sitecore > mediaFramework > playerEventsTriggers` section.
Generators are added via `add` node with three attributes, which include: `name`, `templateId`, `type`. The `name` attribute is a unique identifier among other player event triggers. The `templateId` attribute is the media item templateId created in section 2.1.5. Lastly, the `type` attribute is the fully-qualified type name for the class that implements the `EventTrigger` abstract class.

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

The following code sample demonstrates the registration of events for Ooyala service provider.

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
```

```
          <sitecore>
            <mediaFramework>
              <playerEventsTriggers>
                   <add name="ooyala video"
                        templateId="{C3B8C43D-AD6B-49E2-9C6E-2FB4B53A966A}"
                        type="Sitecore.MediaFramework.Ooyala.Synchronize.VideoEventTrigger,
Sitecore.MediaFramework.Ooyala" />


                   <!-- ... -->

              </playerEventsTriggers>
            </mediaFramework>
          </sitecore>
        </configuration>
```

### 7.2.3 Implement Client Side Event Trigger

To record events, the web browser must issue asynchronous calls during video playback. This is accomplished by calling the JavaScript function `onMediaEvent` and `onMediaChanged` for an instance of `PlayerEventsListener` object during player initialization. `PlayerEventsListener` object is defined in Media Framework JavaScript file path `\sitecore modules\Web\MediaFramework\js\Analytics\PlayerEventsListener.js`.

Developer must create a custom JavaScript file (i.e. `~/JS/<ServiceProvider>.js`), which is referenced in the Media Player Markup (Chapter 6), and is appended to `ScriptUrls` within the

Generate`()` method (see section 6.2.1). The utilization and implementation of the following client side JavaScript methods are required:

#### onMediaEvent()

| | | | |
|---|---|---|---|
| **Method:** | `function onMediaEvent(event, eventType)` | | |
| **Description:** | When service provider video player JavaScript first loads, the client side events must register events via this method. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *event* | `object` | This parameter is specific to the media service provider. This parameter will be utilized when calling other modules within the `PlayerEventsListener.` |
| | *eventType* | `string` | Event type is one of the events specified in the `PlayEventsListener.eventTypes,` which includes: PlaybackStarted, PlaybackCompleted, PlaybackChanged, PlaybackError. |
| **Return value** | | | |
| | **Type:** | `undefined` | |
| | **Description:** | N/A | |

In the following example the `onTemplateReady` is called by the service provider player initializer, which calls `onMediaEvent` and `onMediaChanged`. The `videoPlayer.addEventListener`, `myServiceProviderArg` and `MY_SERVICE_PROVIDER.MediaEvent` variables are specific to the service provider; therefore, the `event` argument is specific to the provider as well.

```
var myCompanyMediaListener = new PlayerEventsListener();

//...

myCompanyMediaListener.onTemplateReady = function (myServiceProviderArg) {

   var videoPlayer = myServiceProviderArg.getVideoPlayer();

   videoPlayer.addEventListener(MY_SERVICE_PROVIDER.MediaEvent.BEGIN
      , function (event) {
      myCompanyMediaListener.onMediaEvent(event, this.eventTypes.PlaybackStarted);
      });

   videoPlayer.addEventListener(MY_SERVICE_PROVIDER.MediaEvent.COMPLETE
      , function (event) {
      myCompanyMediaListener.onMediaEvent(event, this.eventTypes.PlaybackCompleted);
      });

   videoPlayer.addEventListener(MY_SERVICE_PROVIDER.MediaEvent.ERROR
      , function (event) {
      myCompanyMediaListener.onMediaEvent(event, this.eventTypes.PlaybackError);
      });

   videoPlayer.addEventListener(MY SERVICE PROVIDER.MediaEvent.PROGRESS
      , function (event) {
      myCompanyMediaListener.onMediaEvent(event, this.eventTypes.PlaybackChanged);
      });

   videoPlayer.addEventListener(MY_SERVICE_PROVIDER.MediaEvent.CHANGE
   , function (event) {
      myCompanyMediaListener.onMediaChanged(event);
      });
};
```

## onMediaChanged()

| | | | |
|---|---|---|---|
| **Method:** | `function onMediaChanged(event)` | | |
| **Description:** | When service provider video player JavaScript first loads, the client side events must be registered by this method as initialization sequence. This method is called when the video is dynamically changed. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *event* | `object` | This parameter is specific to the media service provider. This parameter will be utilized when calling other modules within the `PlayerEventsListener`. |
| **Return value** | | | |
| | **Type:** | `undefined` | |
| | **Description:** | N/A | |

See `onMediaEvent()` method for an example.

### getMediaId()

| | | | |
|---|---|---|---|
| **Method:** | `getMediaId` | | |
| **Description:** | Corresponds to the unique media id that is generated by the media service provider. This information is specific to the media provider JavaScript API, or JavaScript parameters that was set in the server side `Generate()` method (see section 6.2.1). | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `object` | This parameter is specific to the media service provider, which is passed when the service provider calls onMediaEvent method. |
| **Return value** | | | |
| | **Type:** | `string` | |
| | **Description:** | Media provider video unique Id. | |

```
var myCompanyMediaListener = new PlayerEventsListener();
//...
myCompanyMediaListener.getMediaId = function (args) {
    return args.id;
};
```

### getDuration()

| | | | |
|---|---|---|---|
| **Method:** | `function getDuration(args)` | | |
| **Description:** | Numeric value that represents total duration of the video in seconds.<br>This information is specific to the media provider JavaScript API, or JavaScript parameters that was set in the server side `Generate()` method (see section 6.2.1). | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `object` | This parameter is specific to the media service provider, which is passed when the service provider calls onMediaEvent method. |
| **Return value** | | | |
| | **Type:** | `number` | |
| | **Description:** | A value that represents the video duration in seconds. | |

```
  var myCompanyMediaListener = new PlayerEventsListener();
//...
myCompanyMediaListener.getDuration = function (args) {
    return Math.round(args.duration);
};
```

### getPosition()

| | |
|---|---|
| **Method:** | `function getPosition(args)` |
| **Description:** | Numeric value that represent the current video position (in seconds) with respect to video duration (see `getDuration()`). The functions `getPosition()` and |

| | | | |
|---|---|---|---|
| | | `getDuration()` are used to calculate the percentage of the video completed. This information is specific to the media provider JavaScript API. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `object` | This parameter is specific to the media service provider, which is passed when the service provider calls onMediaEvent method. |
| **Return value** | | | |
| | **Type:** | `number` | |
| | **Description:** | A value that represents the video position in seconds. | |

```
var myCompanyMediaListener = new PlayerEventsListener();
//...
myCompanyMediaListener.getPosition = function (args) {
    return args.position;
};
```

## getContainer()

| | | | |
|---|---|---|---|
| **Method:** | | `function getContainer(args)` | |
| **Description:** | | Html container that encompasses the media item `IFRAME` and Sitecore item parameters. By convention, the media container contains the class name "`mf-player-container`". | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `object` | This parameter is specific to the media service provider, which is passed when the service provider calls onMediaEvent method. |
| **Return value** | | | |
| | **Type:** | `object` | |
| | **Description:** | JavaScript DOM | |

The following code snippet satisfies this requirement, which uses JQuery to find parent element:

```
  var brightcoveListener = new PlayerEventsListener();
//...
brightcoveListener.getContainer = function (args) {
    var id = args.target.experience.id;
    return $('[id="' + id + '"]' ).closest('.mf-player-container');
},
```

## getEventType()

| | |
|---|---|
| **Method:** | `function getEventType(args)` |
| **Description:** | Converts a media service provider event to the respective Sitecore `PlayEventsListener.eventTypes`. This is required only when the second parameter (i.e. `eventType`) of the `onMediaEvent()` method is omitted; thereby, there is a need to extract the event type based on the parameter. |

| Parameters | | | |
|---|---|---|---|
| | **Name** | **Type** | **Description** |
| | *args* | `object` | This parameter is specific to the media service provider, which is passed when the service provider calls onMediaEvent method. |
| **Return value** | | | |
| | **Type:** | `string` | |
| | **Description:** | Event type is one of the events specified in the `PlayEventsListener.eventTypes`, which includes: PlaybackStarted, PlaybackCompleted, PlaybackChanged, PlaybackError. | |

Below example demonstrates Brightcove event mapping.

```
  var brightcoveListener = new PlayerEventsListener();
//...
brightcoveListener.getEventType = function (event) {

    var mediaEvent = brightcove.api.events.MediaEvent;

    switch (event.type) {
        case mediaEvent.PROGRESS:  return this.eventTypes.PlaybackChanged;
        case mediaEvent.BEGIN:     return this.eventTypes.PlaybackStarted;
        case mediaEvent.COMPLETE:  return this.eventTypes.PlaybackCompleted;
        case mediaEvent.ERROR:     return this.eventTypes.PlaybackError;
        default:                   return undefined;
    }
};
```

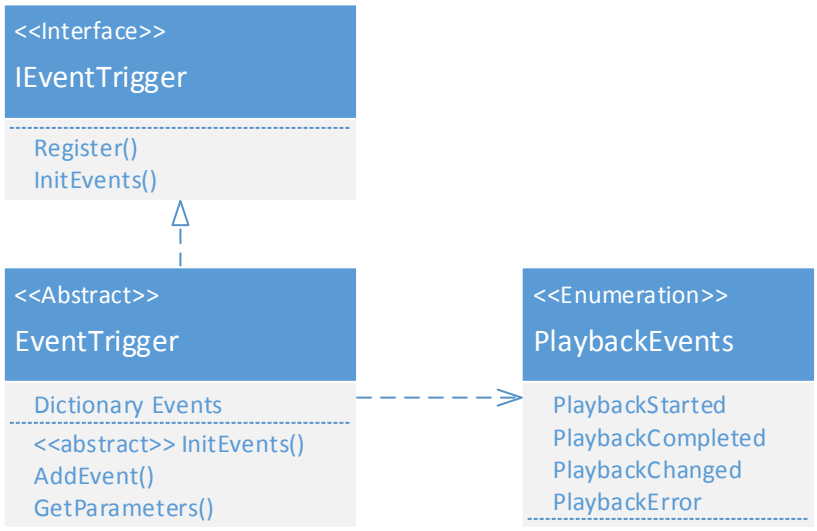
## getAdditionalParameters()

| **Method:** | `getAdditionalParameters` | |
|---|---|---|
| **Description:** | Returns an object that contains `mediaId`, `mediaName`, and `mediaLength`, which are tracked using Sitecore DMS. These fields correspond to properties defined in the Sitecore.MediaFramework.Analytics.EventProperties, which is utilized by your implementation of server side **Error! Reference source not found.** (see section **Error! Reference source not found.**). The abstract `EventTrigger` class passes these data values to Sitecore DMS after the `IEventTrigger.Register()` method is invoked through the pipeline. As a result, you have the flexibility to expand the list of data items that are tracked in Sitecore DMS. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *args* | `object` | This parameter is specific to the media service provider, which is passed when the service provider calls onMediaEvent method. |
| **Return value** | | | |
| | **Type:** | `object` | |
| | **Description:** | An object that contains the following attributes: `mediaId`, `mediaName`, and `mediaLength`. | |

```
        var brightcoveListener = new PlayerEventsListener();
        //...
        myCompanyMediaListener.getAdditionalParameters = function (args) {
            return {
                mediaName: args.title,
                mediaId: this.getMediaId(args),
                mediaLength: this.getDuration(args)
            };
        };
```

**Note**
Prior to recording event triggers, you must deploy Media Framework analytics workflow by selecting Sitecore->Workbox->Analytics Workflow->Deploy All from content editor, and then publishing all the changes.

## 7.3 Class Diagram

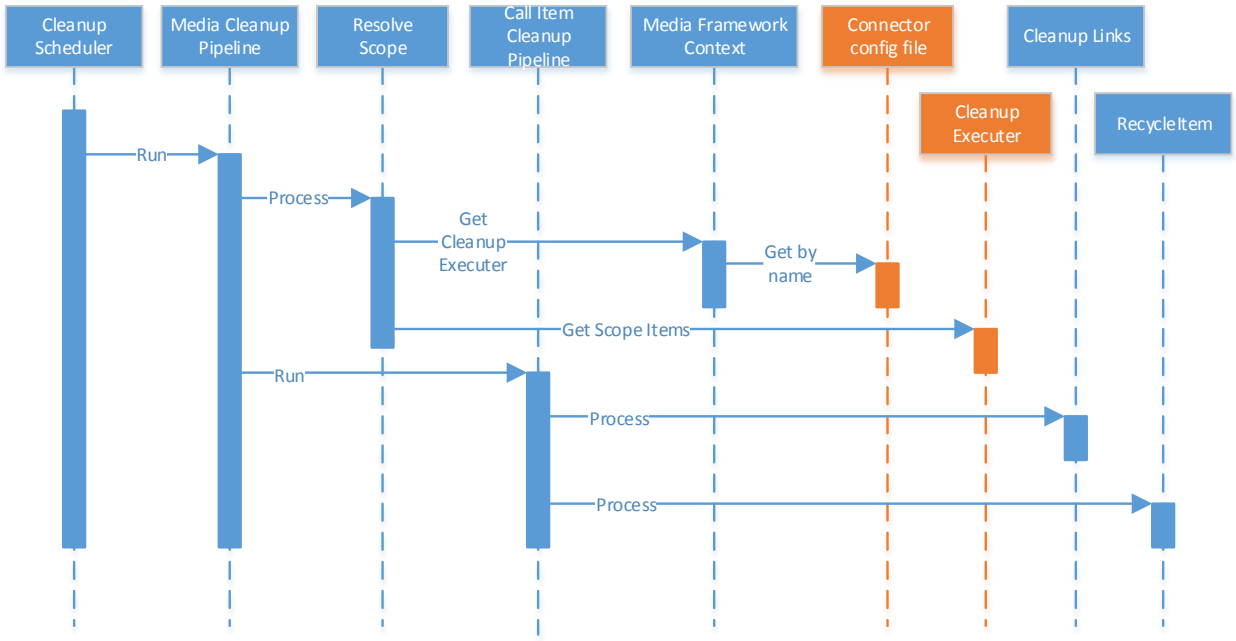


# Chapter 8 Media Cleanup

Media framework cleanup is responsible for removing Sitecore database items that are no longer available through the media service provider.

## 8.1 Overview

The following sequence diagram illustrates the media cleanup process. The orange boxes represent the components that must be implemented. The blue boxes are part of Media Framework, and do not need to be changed.



## 8.2 Cleanup Links

A cleanup executer is responsible for removing links to an item, before the item is removed at a later step by Cleanup Executers.

### 8.2.1 Define Cleanup Links

Cleanup links are responsible for removing links to a Sitecore item before it is deleted. Cleanup links are defined in Sitecore configuration files under `configuration > sitecore > mediaFramework > mediaCleanup > cleanupLinks`.

Cleanup links are added by specifying an `add` node with three attributes, which include: `name`, `templateId`, and `type`. The `name` attribute is a unique identifier among other cleanup links. The `templateId` attribute is the media item templateId created in section 2.1.5. Lastly, the `type` attribute is the fully-qualified link cleanup type, which is set to `Sitecore.MediaFramework.Cleanup.LinkDatabaseCleanupLinks, Sitecore.MediaFramework`.

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

The following code sample demonstrates the cleanup link configuration.

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
            <mediaFramework>
              <mediaCleanup>
                 <cleanupLinks>

                    <add name="mycompany video"
                         templateId="{290942FA-7CBF-4985-B047-E1DAD3A8A975}"
                         type="Sitecore.MediaFramework.Cleanup.LinkDatabaseCleanupLinks,
Sitecore.MediaFramework"/>

                    <add name="mycompany_player"
                         templateId="{70843707-CFF8-4253-AB47-ECE6F3E42229}"
                         type="Sitecore.MediaFramework.Cleanup.LinkDatabaseCleanupLinks,
Sitecore.MediaFramework"/>

                 </cleanupLinks>

              </mediaCleanup>

            </mediaFramework>
          </sitecore>
        </configuration>
```

## 8.3 Cleanup Executers

Cleanup executers are responsible for identifying and deleting Sitecore items that no longer exist on the external media service provider.

### 8.3.1 Implement CleanupExecuterBase<TEntity,TSearchResult>

Cleanup executers inherit from the abstract class `Sitecore.MediaFramework.Cleanup.Sitecore.MediaFramework.Cleanup.CleanupExecuterBase<TEntity, TSearchResult>` where `TSearchResult` is an instance of `MediaSearchResult` for the specific `TEntity` type. `CleanupExecuterBase` abstract class implements `ICleanupExecuter`, which requires the implementation of `GetScopeItems()` to determine Sitecore items that will be deleted. The abstract class `CleanupExecuterBase` has the following properties and methods:

#### IndexName

| | | | |
|---|---|---|---|
| **Property:** | `protected abstract string IndexName {get;}` | | |
| **Description:** | This property corresponds to the content search index name. For example, default Sitecore search utilizes `"sitecore_master_index"`. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *N/A* | `N/A` | N/A |
| **Return value** | | | |
| | **Type:** | `System.String` | |
| | **Description:** | Search provider index name. | |

See Listing 8.1 for an example.

#### ImportName

| | | | |
|---|---|---|---|
| **Property:** | `Public string ImportName {get; set;}` | | |
| **Description:** | This property corresponds to the textual value inside the configuration path configuration > sitecore > mediaFramework > mediaCleanup > cleanupExecuters > add > importName for the respective executer. The `ImportName` property is used to retrieve respective importer so that they could be used to identify Sitecore item that require deletion. For example, section 8.3.2 shows import name `import_mycompany_videos`. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *N/A* | `N/A` | N/A |
| **Return value** | | | |
| | **Type:** | `System.String` | |
| | **Description:** | Returns the name attribute of import executer, which is defined within the configuration file and referenced within the cleanup executer. | |

#### Templates

| | |
|---|---|
| **Property:** | `public List<ID> Templates { get; protected set; }` |
| **Description:** | This property contains list of media template IDs that correspond to the values set in the configuration (see section 8.3.2) path: configuration > sitecore > mediaFramework > mediaCleanup > cleanupExecuters > add > templates > id |

| | | | |
|---|---|---|---|
| | | The template ID reflects the importer created in section 2.1.5 | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *N/A* | `N/A` | N/A |
| **Return value** | | | |
| | **Type:** | `System.Collections.Generic.List<Sitecore.Data.ID>` | |
| | **Description:** | Template ID of the media item (same as importer template) that requires cleanup. | |

## GetEntityId()

| | | | |
|---|---|---|---|
| **Method:** | | `protected abstract string GetEntityId(TEntity entity);` | |
| **Description:** | | Returns the unique Id for the media item that is recognized by the external service provider. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *entity* | `TEntity` | The generic entity type corresponds to the importer(s) specified in section 3.2.2 and 3.2.3 |
| **Return value** | | | |
| | **Type:** | `System.String` | |
| | **Description:** | Unique media Id that is recognized by external media provider. | |

In below example, `MyReadOnlyId` is a unique video id that is recognized by the external media provider.

*Listing 8.1*

```
public class VideoCleanupExecuter
   : CleanupExecuterBase<MyCompany.Entities.Video
                         ,MyCompany.Entities.VideoSearchResult>
{
   protected override string GetEntityId(Video entity)
   {
      return entity.MyReadOnlyId;
   }

   protected override string GetSearchResultId(VideoSearchResult searchResult)
   {
      return searchResult.MyReadOnlyId;
   }

   protected override string IndexName
   {
      get { return "mediaframework_mycompany_index"; }
   }
}
}
```

## GetSearchResultId()

| | | | |
|---|---|---|---|
| **Method:** | | `protected abstract string GetSearchResultId(TSearchResult searchResult);` | |
| **Description:** | | Similar `toGetEntityId()`, this method returns the unique Id for the media item that is recognized by the external service provider. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |

| | | | |
|---|---|---|---|
| | *indexName* | `TSearchResult is-a Sitecore.MediaFramework.Entities.MediaServiceSearchResult` | Sitecore search result entity. |
| **Return value** | | | |
| | **Type:** | `System.String` | |
| | **Description:** | Unique media Id that is recognized by external media provider. | |

See Listing 8.1 for an example.

## AddTemplate()

| | | | |
|---|---|---|---|
| **Method:** | `Public void AddTemplate(string id)` | | |
| **Description:** | This method is used by Sitecore to populate the Templates property for the current class. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *id* | `System.String` | The template ID specified in the configuration file, which reflects the importer created in section 2.1.5. |
| **Return value** | | | |
| | **Type:** | `void` | |
| | **Description:** | N/A | |

## GetServiceData()

| | | | |
|---|---|---|---|
| **Method:** | `protected virtual List<TEntity> GetServiceData(Item accountItem)` | | |
| **Description:** | Using the `ImportName` property, this method utilizes the Import Executer (see section 3.2) to return list of all media elements stored in the external service provider. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |
| | **Type:** | `System.Collections.Generic.List<TEntity>` | |
| | **Description**: | List of generic entity types that correspond to the importer specified in section 3.2.2 and 3.2.3. | |

## GetSitecoreData()

| | | | |
|---|---|---|---|
| **Method:** | `protected virtual List<TSearchResult> GetSitecoreData(string indexName, Item accountItem)` | | |
| **Description:** | Using Sitecore search, returns all data items under the media account. | | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |

| | | | |
|---|---|---|---|
| | *indexName* | `System.String` | Search provider index name, which reflects the `IndexName` property. |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |
| | **Type:** | `System.Collection.Generics<TSearchResult>` Note that `TSearchResult is-a` type of `Sitecore.MediaFramework.Entities.MediaServiceSearchResult` | |
| | **Description**: | Using Sitecore search, returns all data items under the media account. | |

### GetScopeItems()

| | | | |
|---|---|---|---|
| **Method:** | | `public virtual IEnumerable<Item> GetScopeItems(Item accountItem)` | |
| **Description:** | | This method returns a list of Sitecore items that do not exist in the external media service provider. This is the list of Sitecore items will be deleted. | |
| **Parameters** | | | |
| | **Name** | **Type** | **Description** |
| | *accountItem* | `Sitecore.Data.Items.Item` | The Sitecore item that represents the account to use when connecting to the external media service. |
| **Return value** | | | |
| | **Type:** | `System.Collections.Generic.IEnumerable <Item>` | |
| | **Description**: | List of Sitecore items that do not exist on the external media service provider. | |

## 8.3.2 Define Cleanup Executers

The cleanup provider reads cleanup executers from the Sitecore configuration files. Cleanup executers must be defined under the `configuration > sitecore > mediaFramework > mediaCleanup > cleanupExecuters` section.

Each cleanup executer is added using the node `add`, which has two attributes: `name` and `type`. The `name` attribute is a unique identifier among other cleanup executer. The `type` attribute is the fully-qualified type name for the class that implements the cleanup executer. Additionally, the `add` node contains `importName` child node that contains the importer name specified in section 3.2.4. Lastly, the media item template ID from section 2.1.5 is specified.

The following is an example of the configuration for the cleanup executer:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
             <mediaFramework>
                <mediaCleanup>
                   <cleanupExecuters>

                      <add name="cleanup mycompany videos"
                           type="Sitecore.MediaFramework.MyCompany.Cleanup.VideoCleanupExecuter,
Sitecore.MediaFramework.MyCompany">

                          <importName>import_mycompany_videos</importName>
```

```
                        <templates hint="list:AddTemplate">

                            <id>{290942FA-7CBF-4985-B047-E1DAD3A8A975}</id>

                        </templates>

                    </add>

                    <add name="cleanup_mycompany_videos"
                         type="Sitecore.MediaFramework.MyCompany.Cleanup.PlayerCleanupExecuter,
Sitecore.MediaFramework.MyCompany">

                       <importName>import_mycompany_players</importName>

                       <templates hint="list:AddTemplate">

                          <id>{70843707-CFF8-4253-AB47-ECE6F3E42229}</id>

                       </templates>
                    </add>

                </cleanupExecuters>
              </mediaCleanup>

            </mediaFramework>
          </sitecore>
        </configuration>
```

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

### 8.3.3 Define Cleanup Scope

Scope executers are used to point to a list of executers for processing. For example, scheduling an import agent must import media contents like: videos, players, playlists, tags, etc. Scope executor provides a way to reference all importers using a unique identifier, so that it can be referenced via configuration file or Sitecore menu control.

The cleanup scope must be defined under the `configuration > sitecore > mediaFramework > scopeExecuteConfigurations` section.

Cleanup scope is specified using the node `add`, which has two attributes: `name` and `type`. The `name` attribute is a unique identifier among other scope executers. The `type` attribute is the fully-qualified type name to scope execution, `Sitecore.MediaFramework.Scopes.ScopeExecuteConfiguration, Sitecore.MediaFramework`.
Additionally, the `add` node contains two child nodes `accountTemplate` and `scope`. The `accountTemplate` node contains the account template ID. The `scope` node contains child `name` nodes that contain the cleanup executer names.

The following illustrates the cleanup scope execution:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>
             <mediaFramework>
                <scopeExecuteConfiguration>
                   <cleanupExecuters>

                    <add name="cleanup_mycompany_content"
type="Sitecore.MediaFramework.Scopes.ScopeExecuteConfiguration, Sitecore.MediaFramework">
                       <accountTemplate>{1E5A076B-6A9A-4825-9FB1-3E95645352E7}</accountTemplate>
                       <scope hint="list">
```

```
                        <name>cleanup_mycompany_players</name>
                        <name>cleanup mycompany videos</name>
                     </scope>
                  </add>

               </cleanupExecuters>
            </scopeExecuteConfiguration >

         </mediaFramework>
      </sitecore>
   </configuration>
```

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

### 8.3.4 Define Cleanup Schedule

A cleanup scheduling agent must be specified in order to remove deprecated media content from Sitecore database. The agent is defined under `configuration > sitecore > scheduling` section.

Scheduling agent is added using the node `agent`, which has three attributes: `name`, `type`, and `interval`. The `name` attribute is a unique identifier among other scheduling agents. The `type` attribute is the fully-qualified type name to the media framework cleanup scheduler class, `Sitecore.MediaFramework.Scopes.ScopeExecuteConfiguration, Sitecore.MediaFramework`. The `interval` attribute is the sleep duration between every execution.

The `agent` node has two additional nodes to target Sitecore database and to determine the cleanup scope (i.e. set of items to cleanup). The database is specified with the `param` node, which contains the database name. The cleanup scope is specified by simply referencing the scope definition created in section 8.3.3 using an xpath expression.

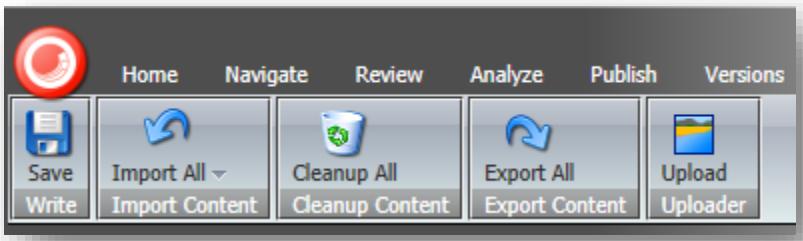
The following example illustrates the cleanup scheduler agent configuration setting:

```
   <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
      <sitecore>
         <scheduling>

            <agent name="MediaFramework Cleanup MyCompany"
                   interval="08:00:00"
                   type="Sitecore.MediaFramework.Schedulers.CleanupScheduler,
Sitecore.MediaFramework">

               <param desc="database">master</param>

               <scopeConfigurations hint="raw:AddConfiguration">

                  <add ref="mediaFramework/scopeExecuteConfigurations
/*[@name='cleanup_mycompany_content'][1]"/>

               </scopeConfigurations>

            </agent>

         </scheduling>
      </sitecore>
   </configuration>
```

**Note:**
You should create a configuration file specifically for your connector. Do not add settings directly to any of the configuration files included in Media Framework.

## 8.4 Cleanup Command

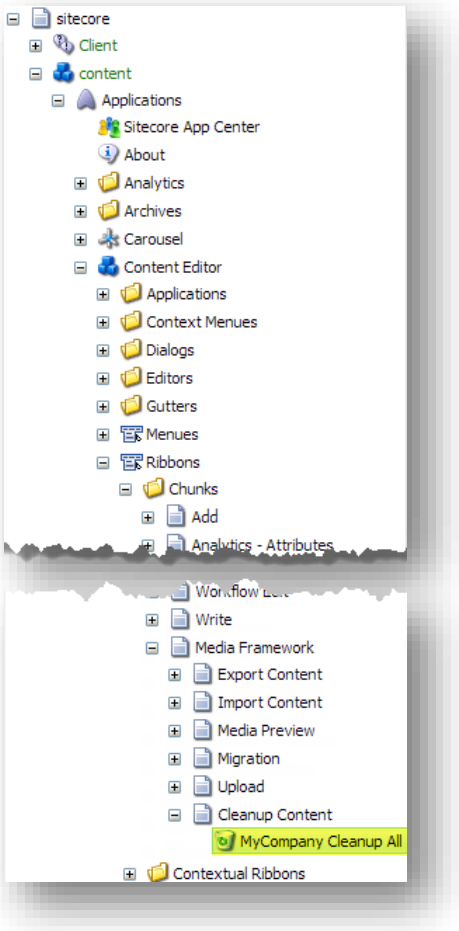
The cleanup process can be started manually within the Sitecore client. This section describes how to add this functionality to your connector.



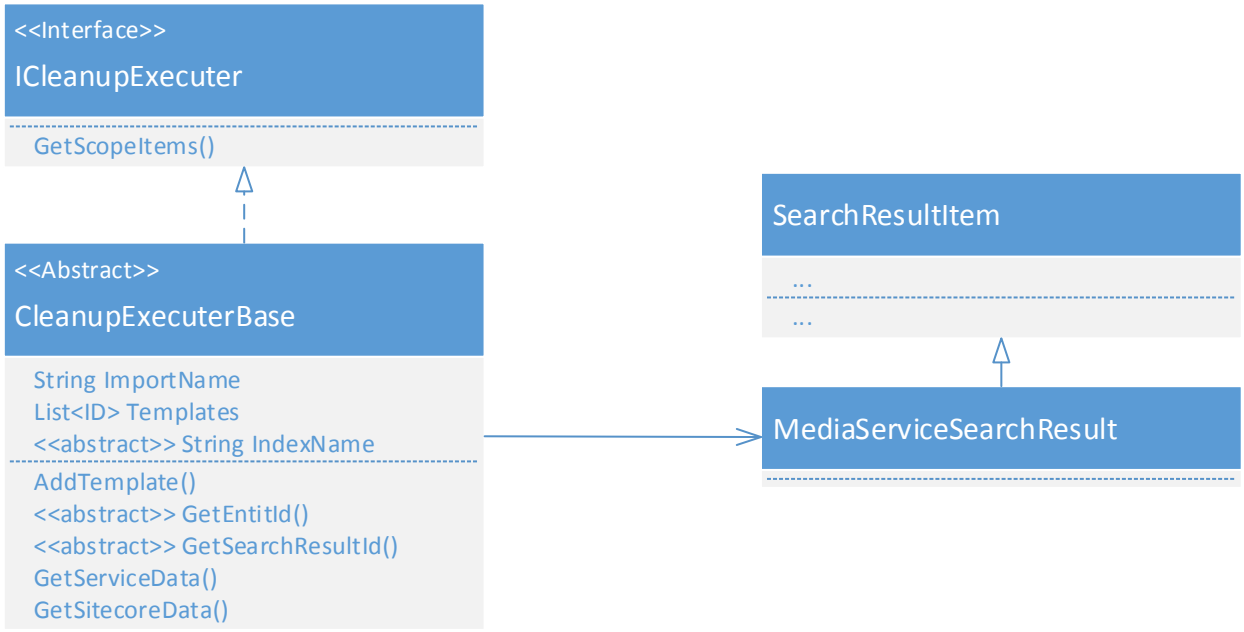

### 8.4.1 Add the Manual Cleanup Button

Media Framework provides the ability to cleanup all media for an account.

1. Open Content Editor.
2. Switch to the **core** database.
3. Navigate to `/sitecore/content/Applications/Content Editor/Ribbons/Chunks/Media Framework/Cleanup Content`
4. Create a new item using the following settings:
    a. **Name:** <ProviderName> Cleanup All
    b. **Template:** `/sitecore/templates/System /Ribbon/Large Menu Combo Button`
5. Set the following field value:
    a. **Field name:** Header
    b. **Value:** Cleanup All
6. Set the following field value:
    a. **Field name:** Icon
    b. **Value:** Applications/32x32/garbage_empty.png
7. Set the following field value:
    a. **Field name:** Click
    b. **Value:** mediaFramework:ManualCleanup:Brightcove (scope=`cleanup_mycompany_content`)
8. Set the following field value:
    a. **Field name:** Tooltip
    b. **Value:** Remove all items that not exist in the media service



## 8.5 Class Diagrams



# Chapter 9 Index