Platform, Services, and Utilities

Generated on: 2024-12-03 10:59:44 GMT+0000

SAP Commerce | 2205

Public

Original content: https://help.sap.com/docs/SAP_COMMERCE/d0224eca81e249cb821f2cdf45a82ace?locale=en-US&state=PRODUCTION&version=2205

mediaconversion Extension

The `mediaconversion` extension provides a framework to convert media of one format to media of another format utilizing the Media Container model.

To do this, it features a special kind of media format including conversion information and strategy. Furthermore, it hosts an implementation to scale, convert, and modify bitmap image media using the ImageMagick open-source image manipulation toolkit.

Note

The `mediaconversion` extension is disabled by default. If you want to enable it, you have to add the following entry to the content of <extensions> element in your **`localextensions.xml`** file:

```
<extension dir="${HYBRIS_BIN_DIR}/modules/platform/mediaconversion"/>
```

Then you have to build and initialize your system. Find more information on building in the Installation Based on Specified Extensions document.

Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Media Conversion Overview

SAP Commerce data management is rapidly evolving into a centralized repository for all sorts of data and serves as a backbone for all channel publishing activities. Since each channel has different media asset requirements in terms of quality or resolution, it is mandatory that SAP provides media conversion functionality.

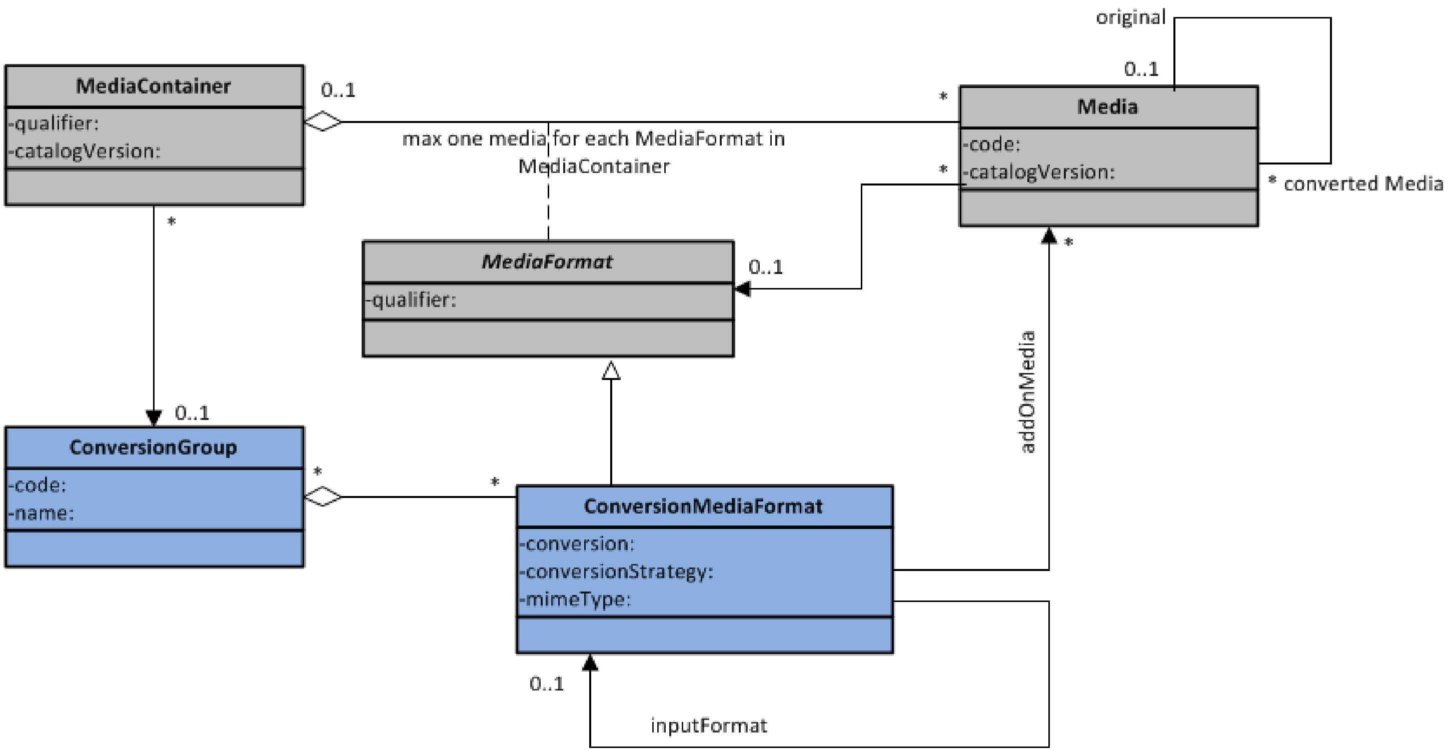
Media Conversion Concept

The idea of a Media Container aggregating multiple representations of one and the same logical media asset already existing in the SAP system. The `mediaconversion` extension adds a framework to automatically generate derived media asset. In other words, to convert an input to a desired output media.

Data Model

To aggregate media of the same kind but in different formats, the MediaContainerModel is leveraged. With this, a MediaContainerModel can be seen as a logical media having several physical representations, for example, medias. Within a MediaContainerModel, there can be one representation for each MediaFormat in the system retaining the original image from which other MediaModel is derived or converted from.

This original media, also referred to as **master**, has a special role. It is identified by the media within the MediaContainerFormat that has neither the original nor the originalDataPK attribute set. These two attributes are manageable information stored by the `mediaconversion` extension. It is used to track the conversion paths and to detect changes and required reconversion. As illustrated, upon conversion the newly created or updated MediaModel gets a reference to its original media set, `original` attribute. The originalDataPK is set representing the exact version of the conversion input. The conversion is also responsible for setting the derived mediaContainer, catalogVersion, and mediaFormat accordingly.



Defining the Media Master

When discussing conversion, it is necessary to understand which media is chosen as the initial source of all conversions. The `mediaconversion` extension mechanism is used to pick the one media within a MediaContainer that does not have the `original` nor the `originalDataPK` attribute set, for example, is NULL. If there is no such media or there are multiple medias matching this condition within a MediaContainer, the master media of this will not be determined and the MediaContainer is treated accordingly as `emtpy` or `not convertible`. These two attributes are considered as administrative data of the `mediaconversion` extension and should not be modified. Also, any modifications on MediaContainer contents of containers containing converted medias is not recommended.

The source media for a particular conversion could be a different media as described in the Conversion Chaining section below.

i Note

Manual Changes

Users should not manually delete converted medias from their media containers without deleting them entirely from the database. All subsequent conversion attempts fail because of unique constraint violations, same generated code, and catalogVersion.

Conversion Media Format

Platform. The ConversionMediaFormat represents a physical format in that context. The MediaFormat instance can still be used to represent logical format which is mapped to a physical one by a MediaContext. The ConversionMediaFormat introduced with the new ConversionMediaFormat extends the MediaFormat and denotes a format that can be converted by the `mediaconversion`.

The SAP Commerce can be shipped with the sample data of ConversionMediaFormat. Find description in the Conversion Media Formats - Sample Data document.

Conversion Command and Strategy

The ConversionMediaFormat or one of its subtypes contains all the information required to convert a MediaModel into a new format. The main attributes to do so are:

- conversionStrategy: An attribute that contains the Spring bean ID of the MediaConversionStrategy to use. This bean must be available in the Spring context.
- conversion: An attribute that contains the conversion instructions. This attribute is dependent on the MediaConversionStrategy used.

  To follow examples of bitmap image conversions, see the *ImageMagick Usage* section of ImageMagick Configuration.
- Mime Type of the output: Optional. It can be set depending on the conversionStrategy used . If no mime type is provided, the mime type of the media input is used.

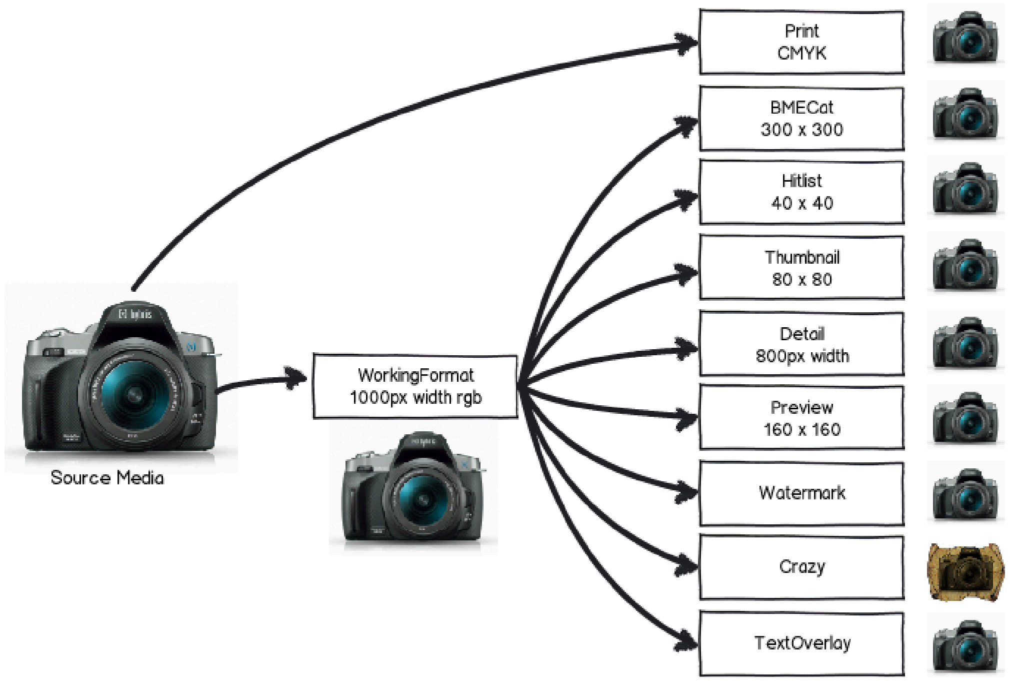
Additional Medias Used in the Conversion

As some conversions need additional input besides the input media, the ConversionMediaFormat hosts a list of mediaAddOns. For example, you may want to put a watermark image over an image or apply a special color profile in the color space transformation for a print production. Having an attached XSLT stylesheet, it is also possible to convert XML documents from one format to another. The interpretation and usage of additional media is up to the MediaConversionStrategy implementation.

Conversion Chaining

The ConversionMediaFormat can be chained. For example, a ConversionMediaFormat can restrict its input media to be of a certain ConversionMediaFormat. This is important as some conversion operations are consuming procedures and should not be redone for each target format. When you have very large input images are very large, it does not make sense to derive all web frontend images from this huge binary data. A working format could be defined which is then reused as a conversion input. With this, it is also possible to build complete conversion pipelines.

The following diagram shows such scenario for images:



Conversion Group

Not all input media are supported or appropriate for every converted media format. For example, your banner or promotion images need to be converted to completely different formats than your product detail pictures. You have technical documentation such as XML assigned to your product that must be converted, but in a different way than your images. To reflect this, the `mediaconversion` extension introduces the concept of a ConversionGroup, which is a simple aggregation of target conversion formats attached to the MediaContainer. If this optional attribute is set on the MediaContainer, all conversion mechanisms are converted to the referenced ConversionMediaFormat, including parent formats in the conversion chain.

Specifying a ConversionGroup is optional. If none is set, all ConversionMediaFormats in the system are considered as target formats. If you wrap your product pictures in a MediaContainer and want to leverage conversion within SAP Commerce for any asset, you need to consider ConversionGroups.

Conversion Status

The `mediaconversion` extension exposes a dynamic attribute to monitor the current conversion status of a MediaContainer. The states are defined as follows:

Status	Description
CONVERTED	There is a defined master media and all conversion formats (of the associated ConversionGroup or in general) have a corresponding representation in this MediaContainer.
EMPTY	There is no identified master media for this MediaContainer. This is either because there are no medias associated to this MediaContainer or no media has the `original` or the `originalDataPK` attribute set.
PARTIALLY_CONVERTED	There is a defined master media and some converted media present in this MediaContainer, but there are still conversion formats (either in the associated ConversionGroup or in general) which do not have a representation in the MediaContainer.

Status	Description
UNCONVERTED	There is a defined master media but no converted medias in this MediaContainer.

Conversion Error Log

If a conversion failed, the `mediaconversion` extension generates a ConversionErrorLog entry within the database that contains the fault information. These entries are automatically deleted upon successful conversion of the media to the specified format. The latter behavior can be turned off, for example the ConversionErrorLog is kept forever by adjusting the `mediaconversion.removeConversionErrorLogUponSuccess` property to false.

Conversion Mechanisms

Spring Service

The new MediaConversionService exposes the functionality to convert media within the MediaContainer to different formats declared in the ConversionMediaFormat. The MediaConversionService functionality to retrieve media in a specify format, `getMediaByFormat()`, remains unchanged and queries the database for a matching media. It throws a ModelNotFoundException if no such media exists. It is important to note that the MediaContainer becomes the central item for automatic conversion. The media must reside in a MediaContainer with a marked master, for example, media of format `original`, to be converted by the provided service or cron jobs. The targeted format must be a ConversionMediaFormat instance.

Media Conversion Cron Job

In addition to the above-mentioned MediaConversionService that supports explicit conversion calls, the `mediaconversion` extension comes with a MediaConversionCronJob. The MediaConversionCronJob queries the database for all missing media in any MediaContainer and triggers the conversion accordingly.

The MediaConversionCronJob supports the following configuration options:

- maxThreads: The number of threads to use for conversion. Setting this value higher than the limit of the **processExecutor** is counterproductive.
- catalogVersion: Restricts the examined MediaContainer to a certain catalogVersion.
- includedFormats: Specifies a list of ConversionMediaFormat which are to be converted. If this property is not set, all ConversionMediaFormat in the system are converted.
- asynchronous: If set to `true`, the MediaConversionCronJob only creates a task instance to make the conversion—one for each MediaContainer for which some action needed. With this, you can leverage the whole cluster to make conversions, but the tasks created are not restricted to a specific cluster node.

A MediaConversionCronJob is used to generate derived medias: one to extract media metadata and one to delete them. They are set up in the `mediaconversion` extension project data upon initialization or update. The cron jobs are deactivated and serve as a template for further customization.

Media Conversion Task Runner

Additionally, there is a MediaConversionTaskRunner that leverages asynchronous media conversion with the help of the Task Engine. This implementation is used by the MediaConversionCronJob with set `asynchronous` configuration option. Although the MediaConversionTaskRunner accepts standard TaskModel instances for processing, it expects a MediaConversionTaskContext instance set as the `TaskContext` attribute.

On Demand Conversion

Another integrated way to convert media to different formats is called `on demand` or `on the fly` conversion. For example, converting the media to the desired format when they are first accessed like on a website. This feature is useful in scenarios in which medias are uploaded and updated very frequently.

Flow

When converting medias on the fly on a website, it is important to separate the HTML rendering request from the request that makes the conversion. As shown in the following figure, the complete flow is separated into three phases:

1. Upon the page request (1) the controller, servlet, or JSP checks for the availability of the media in the required format (2). If such a media exists, its URL is returned as usual. If not, a substitute URL is embedded in the HTML page and returned (3).
2. The client browser requests the media from the substitute URL (4). A simple servlet makes the conversion (5) and sends back a redirect to the URL of the newly created image (6).
3. The client requests the media (7) and retrieves it (8). Note that the media delivery through a HTTP Deamon is still possible.

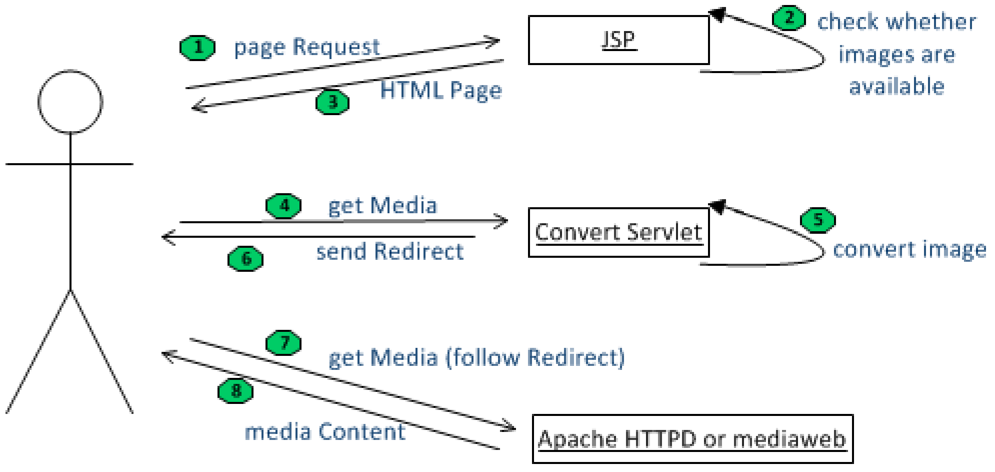


Figure: The sample process of on demand conversion.

Convert Servlet

This feature is already incorporated in the `mediaconversion` extension. Therefore the `mediaconversion` extension exposes a servlet, `/mediaconversion/convert`, that handles the appropriate substitute URLs. You need to expose and secure the conversion servlet accordingly. The accepted URL is `/mediaconversion/convert/` ***PK of MediaContainer /qualifier of ConversionMediaFormat***.

Tag Library

To make it easier to leverage on-demand conversion, the `mediaconversion` extension comes with a little tag library (taglib) to include task described in step 3 in the Flow section. The taglib URL is **`http://www.hybris.de/jsp/mediaconversion`**.

Enabling the Taglib

To enable the taglib, you have to copy over the taglib descriptor `/mediaconversion/resources/mediaconversion/taglib/mediaconversion.tld` to your web application `WEB-INF/tld` directory. You can automize this by adding the following snippet to your web extension `buildcallbacks.xml` file. To do so, you need to replace all ***yourextension*** occurrences in the code with your extension name

```
<macrodef name="yourextension_before_compile_web">
        <sequential>
                <mkdir dir="${ext.yourextension.path}/web/webroot/WEB-INF/tld" />
                <copy file="${ext.mediaconversion.path}/resources/mediaconversion/taglib/mediaconve
                        tofile="${ext.yourextension.path}/web/webroot/WEB-INF/tld/mediaconversion.t
        </sequential>
</macrodef>
```

URL Tag

URL tag is a tag to output or set a variable to the URL of a media in a given format. The URL generated either points directly to the media or, if the specified format has to be generated first, to the `mediaconversion` extension servlet that makes conversion in a different thread.

The following table describes the attributes:

Name	Description	Required	Runtime Expression	Type
container	The container in which the media in question resides.	true	true	MediaContainerModel
format	The qualifier of the output format (MediaFormatModel). If no format is specified or the specified format is the empty string, the master media of the container is used.	false	true	java.lang.String
scope	The scope of the variable to set. One of page, request, session, or application.	false	false	java.lang.String
var	The name of the variable to set.	false	false	java.lang.String

HTML Image Tag

HTML Image Tag renders a HTML `<img>` element that references a (converted) media in a specified format. The generated URL either points directly to the media or, if the specified format has to be generated first, to the mediaconversion convert servlet that makes conversion in a different thread.

The following table describes the attributes:

Name	Description	Required	Runtime Expression	Type
align	Optional attribute for the generated HTML `<img>` element. The possible values for this attribute are top, middle, bottom, left, and right.	false	true	java.lang.String
alt	Optional alternative text to override alternative text set on the media.	false	true	java.lang.String
border	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
container	The container to output.	true	true	MediaContainerModel
cssClass	Optional CSS class to be used.	false	true	java.lang.String
format	The qualifier of the output format (MediaFormatModel). If no format is specified or the specified format is the empty string, the master media of the container is addressed.	false	true	java.lang.String
height	Optional height of the generated `<img>` element. This height is not obeyed in an image scaling, for example it is only used in the generated HTML output.	false	true	java.lang.String
id	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
ismap	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.Boolean
longdesc	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
name	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
style	Optional (CSS) style attribute.	false	true	java.lang.String
title	Optional attribute for the generated HTML `<img>`	false	true	java.lang.String

Name	Description	Required	Runtime Expression	Type
	element.			
usemap	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
vspace	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
width	Optional width of the generated `<img>` element. This width is not obeyed in an image scaling, for example it is only used in the generated HTML output.	false	true	java.lang.String

Media MetaData Extraction

The `mediaconversion` extension furthermore hosts an extensible mechanism to extract metadata from arbitrary medias. It can be triggered within Backoffice or by cron job. The `mediaconversion` extension comes with an implementation for bitmap image data based on ImageMagick. It is used to extract various information stored in the image itself such as dimensions, color space, Exif data.

The metadata extraction can be easily extended by implementing the MediaMetaDataProvider interface and registering the implementation in the Spring context. All implementations of MediaMetaDataProvider are picked up and called. It is important to restrict implementation to the applicable mime types.

Related Information

The Task Service
ImageMagick Configuration
Media Conversion
ImageMagick - Integration Guide
task Extension - Technical Guide