

# Proposed changes to Rygel for support of pre-encoded and device-sourced content

## *v0.2, The CableLabs CVP-2 reference server team*

### Introduction/background

Today the Rygel DMS (librygel-server) excels at exposing "user-supplied content." That is it supports a model where a user has a collection of content that they store/upload to their system and make available to other UPnP/DLNA devices. To support this, Rygel - via the Tracker and MediaExport plugins - has the ability to extract metadata from the content and make it discoverable and browsable by other DLNA devices/software. And Rygel has the ability to stream the content using loadable MediaEngines - supporting byte-based seeking for the user's content and optional software transcode of some content (with time-based seek).

There is another use case for content - one that we are trying to enable: "device-supplied content." This is a class of content where a device itself is the originator/manager of the content - or has special knowledge and/or capabilities related to the content.

There are a few aspects of this form of content that are different than the user-supplied content model.

- **Device-supplied content may not be stored as simple/single flat files.**
  - For instance, a device may produce multiple files containing the same content in different formats, resolutions, bitrates, etc. e.g. a stereoscopic camera might produce a stereo image file and standard jpeg file each time the user takes a picture. Or a camcorder may split recordings into multiple files to keep file sizes down.
  - A device such as a DVR may store a recording on a non-general-purpose filesystem or in a series of files with an index file(s) providing navigational information.
- **Device-supplied content metadata may not be stored in the content itself.**
  - For instance, a device can have higher-level metadata that can't be stored in the content. e.g. a DVR will often know the title of the show it is recording, what time it was recorded, the content/container format it is in, a description, actors, etc. Most recording content formats don't have the capability to store this information within the content.
  - Content metadata may be stored separately from the content in a device-specific format. e.g. MythTV maintains a SQL database containing almost all recording metadata - all the way down to time/byte offset information for the recording (frame indexes).
- **A device may have special capabilities/requirements for streaming/transforming/link-protecting device-supplied content.**

- For instance, a device may have the ability to allow secondary content forms (hardware transformations) and allow certain forms of content navigation based on its built-in metadata.
  - e.g. Many commercial DVRs contain hardware transcoders that can present content recorded as MPEG2 in various forms of ISO H.264 that can be viewed/transferred to mobile devices.
  - e.g. The MythTV DVR supports time-based content seeking via its content database.
- There may be a requirement to prevent direct access to recorded content via the network and transfer content using link-protected navigation methods.
  - e.g. Copy-protected content may need to be link-protected using DTCP to be viewed on other devices. This affects both the exposed content format(s) and how the content can be navigated. And the content protection may be applied in a tamper-resistant module.

The goal of the changes we're proposing for incorporation into Rygel is to (a) expand the base librygel-server infrastructure to support both the user-supplied and device-supplied/device-managed content models and (b) to improve Rygel's overall DLNA compliance. This will allow the Rygel DMS to be more easily adapted to a variety of devices. And it will support our use case for DLNA CVP2 conformance testing.

While this introduces a few sweeping changes to the Rygel server architecture - and a correspondingly set of code changes - we were mindful of the existing patterns and conventions in the Rygel code. And we believe that while adding new functionality, all the existing functionality of librygel-server and the MediaServers/MediaEngines distributed with Rygel has been retained.

## Change Details

This section is intended to provide more detailed goals/justifications for our changes to librygel-server. This will hopefully help with understanding and justification of the particular code changes.

### 1. The MediaEngine is the source of record for all streamed MediaItem resource metadata.

- **References:**
  - class MediaResource (new)
  - MediaEngine.get\_resources\_for\_item (MediaObject item)
  - plugin MediaExport (changed to populate the MediaObject resource list via MediaEngine.get\_resources\_for\_item())
  - plugin Tracker (changed to populate the MediaObject resource list via MediaEngine.get\_resources\_for\_item())
- **Reasons for the change:**
  - We needed to enable PlaySpeed and DTCP support. Both of these affect the resource metadata and binary stream produced for the “primary” resource.

- The existing Rygel MediaItem/MediaServer derives the primary resource metadata by introspecting the content directly, which may not be possible or necessary for many device-/content-specific use-cases.
- Our content ingest tools generate separate files for scaled-rate content. i.e. 4x and -4x content is stored in separate content files. This is also the direction DLNA is headed with scaled reference content.
- We needed to enable time-indexed content, utilizing index files, for the primary resource of many formats. So the primary resource metadata needs to indicate time-based seek capability for these content formats.
- For supporting CVP2 client testing, we needed to be able to expose content at particular known DLNA profiles for DLNA-supplied reference content, in these cases the profile is predetermined. This is also the use-case for device-sourced content as well.
- Utilizing DIDLLiteResource to represent resource metadata returned by MediaEngine.get\_resources\_for\_item() proved to not be practical as
  - DIDLLiteResource can only be created via a DIDLLiteItem. And engine resource extraction needs to be supported outside the context of didl-lite generation (e.g. for populating an item database)
  - DIDLLiteResource is geared for DIDL-lite xml generation and requires fields to be set that aren't known to the MediaEngine (e.g. the uri)
  - It proved useful to add fields not used for didl-lite generation into Rygel's resource representation (e.g. "name" and "extension" are not part of res block metadata, but are important internally)

So the MediaResource classes is added to represent UPnP item resources in the same way and for many of the same reasons that MediaItem represents UPnP items.

- **Secondary effects of the change:**

- Transformations/TransformationManager functionality seemed redundant once get\_resources\_for\_item() was added to the MediaEngine API. The GstMediaEngine transformation logic was adapted to this method allowing TransformationManager and related logic to be removed.
- Other logic that required resource metadata to be generated exclusively inside the context of a DIDLLiteItem is simplified to just return a MediaResource.
- MediaServers have the option to populate their database with all resource representations returned from the MediaEngine. This in turn can enable CDS Search implementations which leverage database functionality to match non-primary resource metadata.
  - e.g. A CDS Search for items including a DLNA profile can match items who have the profile available via transcoding or as a secondary form.

## **2. The DataSource class is changed to accommodate DLNA PlaySpeed and enable response header feedback prior to streaming**

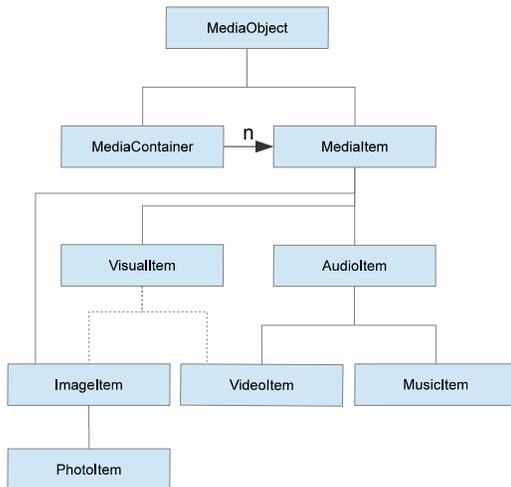
- **References:**

- DataSource.preroll (HTTPSeekRequest? seek, DLNAPlaySpeedRequest? speed): Returns a list of HTTPResponseElements (new)
- DataSource.start () (changed to take no params - params now supplied via preroll())
- class HTTPSeekRequest (modified to remove range start/end values)
  - subclass HTTPByteSeekRequest (modified from "HTTPByteSeek")
  - subclass HTTPTimeSeekRequest (modified from "HTTPTimeSeek")
  - subclass DTCPClearTextByteSeekRequest (new)
- class DLNAPlaySpeedRequest (new)
- class HTTPResponseElement (new)
  - subclass HTTPByteSeekResponse (new)
  - subclass HTTPTimeSeekResponse (new)
  - subclass DLNAPlaySpeedResponse (new)
  - subclass DTCPClearTextByteSeekResponse (new)
- **Reasons for the change:**
  - We needed to enable PlaySpeed support for DLNA CVP-2 compliance testing.
    - The DLNAPlaySpeed object is created in response to a PlaySpeed.dlna.org request header and communicates the request speed to the MediaEngine.
  - A number of DLNA requests require a response header that reflects details of the streamed content - including cases where only a HEAD request is received. Since DataSource.start() already had the concept of processing a seek request, the DataSource is modified to (a) allow seek/speed to be passed to the DataSource prior to streaming (via preroll()), and (b) allow the DataSource to return responses (in the form of HTTPResponseElements).
    - e.g. A TimeSeek response must include the effective time range returned (accounting for decoder-friendly access points and rate-conditional bounds checks) and - when byte-seek is supported - the corresponding byte range.
  - The requirements for DLNA request processing for DTCP-encoded content require the server to return the Content-Range.dtcp.com response header for certain requests
    - As this header contains information that only the MediaEngine should be expected to know (correspondence between encrypted and unencrypted data blocks in this case), the preroll() allows for a DTCPClearTextByteSeekResponse to be returned that is unrelated to the request.
    - Allowing for just mutation of the passed seek/speed request parameters would not accommodate this use case since this header must be returned in a variety of cases - including allowing for response header generation without a seek/speed request.

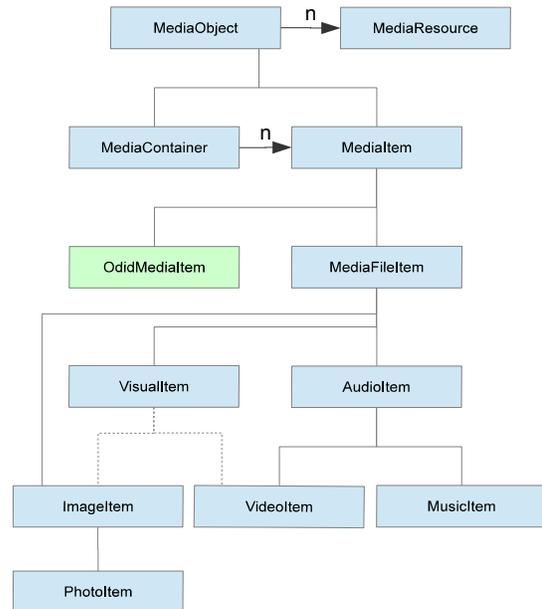
**3. A new MediaFileItem subclass is introduced. This class represents MediaItems which represent user-supplied, uploaded, or externally-hosted content**

- **References:**

- class MediaObject (changed to contain a MediaResource list and related resource serialization logic)
  - subclass MediaContainer (changed to add playlist resources to the MediaObject's MediaResource list)
  - subclass class MediaItem (changed to remove logic specific to/assuming file-accessible content - allows subclasses to have device-sourced metadata/content)
    - subclass MediaFileItem (changed to embody all logic assuming file-accessible content and resource production rules that operate on the MediaObject resource list)
      - subclass AudioItem (changed resource serialization logic)
        - subclass VideoItem (changed resource serialization logic)
        - subclass MusicItem (changed resource serialization logic)
      - subclass VisualItem (changed resource serialization logic)
      - subclass ImageItem (changed resource serialization logic)
- class Thumbnail (changed to return a MediaResource)
- multiple classes had references to MediaItem changed to MediaFileItem



Existing MediaObject Class Hierarchy



Updated MediaObject Class Hierarchy

- **Reasons for the change:**

- The content we utilize to support time-seek and rate-adjusted streams isn't contained in a single file

- The protocol-info attributes and profile are known for DLNA reference content (and device-sourced content in general)
- In DTCP scenarios, the source content is not directly exposed/accessible. In other words, the primary resource representation is effectively a transformation of the source content.
- We wanted to utilize the existing MediaObject hierarchy for our MediaServer (we didn't want to create an entirely separate MediaObject-like tree). While there are differences in models, there is still a lot in common.
- We wanted to maintain the functionality of the existing MediaExport and Tracker MediaServers without making larger changes.
  - MediaFileItem allows subclasses and existing MediaServers to retain their model and assumptions about file-based access, metadata extraction, and internalized resource access.
  - Some systems with device-sourced content may still want to support user-supplied content/content upload.
  - Changes were limited to the addition of logic to add the resources to the MediaObject's MediaResource list - minimizing the changes to MediaExport/Tracker.

#### **4. HTTP request processing needed to be changed to remove assumptions about file-based resources, add support for PlaySpeed, add support for DTCP Range requests, and improve overall DLNA compliance**

- **References:**

- class HTTPGetHandler (added delegation for size, duration, seek availability, and transfer modes)
  - subclass HTTPMediaResourceHandler (new)
  - subclass HTTPIdentityHandler (removed - thumbnail/subtitle handlers added)
  - subclass HTTPThumbnailHandler (added)
  - subclass HTTPSubtitleHandler (added)
  - subclass HTTPTransformationHandler (removed)
  - subclass HTTPPlaylistHandler (removed)
- class HTTPRequest (expanded request processing logic)

- **Reasons for the change:**

- The HTTP processing logic needed to have logic/assumptions regarding file-based access to content isolated so we could support multiple resources per item and multiple files per resource (without creating fake transcoders).
- Requirements regarding the handling of the DLNA transferMode needed improvement to testing.

- Assumptions regarding the content length of the primary resource needed to be adjusted to not assume that the size of the user-supplied/raw file (the MediaItem "size" field) was the assumed size of the resource since content may be (a) not file-accessible, (b) may be transformed (e.g. DTCP encapsulated/encrypted at runtime)
- Assumptions regarding the ability to support time-seek and byte-seek needed to be relaxed/delegated to support a variety of seek types on primary and secondary resources.
- HEAD request processing and overall Response header generation needed to be expanded for DLNA compliance.
- Range.dtcp.com request/response processing needed to be added.
- PlaySpeed.dlna.org request/response processing needed to be added.
- **Secondary effects of the change:**
  - The HTTPContentURI logic was greatly simplified.
  - A number of TODOs and class-specific logic ("if (object is class)...") logic was replaced with HTTPGetHandler delegation calls that were added to meet the above requirements.

## Potential Enhancements

1. MediaExport could be enhanced to store all resource representations in its database for improved CDS SEARCH capability and to improve efficiency.
2. There's potential for more consolidation of resources types, reducing the need for HTTPGetHandler delegation and URI forms.
3. It's conceivable that a device may want to support both user-supplied/uploaded content and device-sourced/managed content. It would be useful in these cases to have the ability to enable multiple MediaServers and MediaEngines.
4. It would be beneficial to have simple containers available for UPnP/DLNA-defined item and container metadata defined and maintained via gupnp or gupnp-av so that Rygel's MediaItem/MediaResource didn't have to re-define (and maintain) these common fields. The gupnp-av DIDLiteItem, DIDLiteResource, and ProtocolInfo classes aren't well-suited for use outside serialization.
5. DataSource.preroll () should probably be async, to accommodate MediaEngines that have to perform IO or potentially time-intensive operations to calculate time/data offsets, encryption overhead, etc. (although this necessarily should be limited to support low-latency HEAD request performance)
6. We didn't give rtp delivery any attention.
7. Support for limited/live operation modes was not fully accounted for. More work is required here for sure.
8. Support for NPT chunk headers when streaming scaled (PlaySpeed) content needs to be added (DLNA 7.5.4.3.3.17)
9. Support for ChunkEncodingMode.dlna.org needs to be added
10. The GstMediaEngine should be returning the effective time range for a TimeSeekRangeResponse when transcoding.

11. The ability to have more complex forms of content representations (which these changes enable) opens up the possibility for opportunistic indexing and pre-transcoding of user content (similar to what the Plex (tm) media server does). This can allow for (a) time seek navigation of user content, (b) a better user experience for transcoded content (by not requiring realtime transcode capability) and (c) improved interoperability (since pre-transcoded streams can support both byte- and time-based seek with full random access)