# hls2

libhls2 is an attempt to modernize & re-write the existing iWedia HLS library: <https://gerrit.iwedia.com/admin/projects/libhls> (master branch)

All the principles applied to **libdash(**[**https://gerrit.iwedia.com/admin/projects/libdash**](https://gerrit.iwedia.com/admin/projects/libdash)**)**, **libiwu(** [**https://gerrit.iwedia.com/admin/projects/libiwu**](https://gerrit.iwedia.com/admin/projects/libiwu)) and **libiwp(**[**https://gerrit.iwedia.com/admin/projects/libiwp**](https://gerrit.iwedia.com/admin/projects/libiwp)**)** libraries are applied here.

Current libhls2 implementation follows the latest HLS draft available at this time: **rfc8216bis-06** (<https://tools.ietf.org/pdf/draft-pantos-hls-rfc8216bis-06.pdf>)

From the top level, the library consists of the following packages:

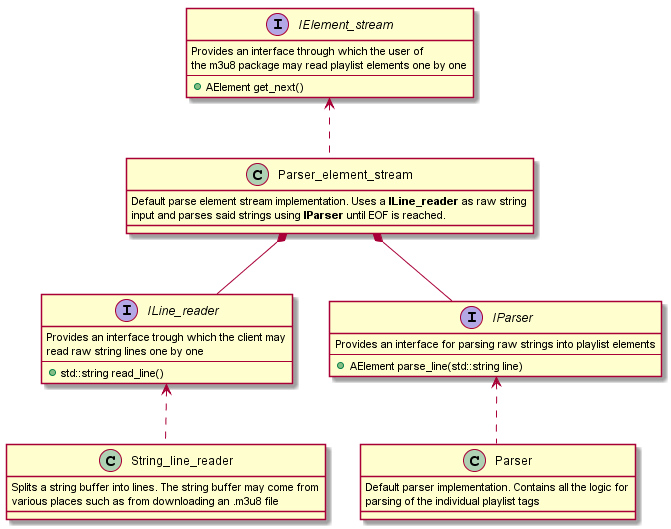
1. **M3U8** - Package in charge of parsing raw manifests and their tags
2. **Playlist** - Package in charge of receiving parsed M3U8 elements and organizing them into media and master playlists
3. **Core** - Package in charge of working with playlists, in order to open/close streams as well as download individual segments

## m3u8 package

The m3u8 package works with raw string input. Where the input comes from is not the responsibility of this package (e.g. it may be downloaded, or read from a buffer elsewhere)

### Top level interface

Figure below illustrates the top level API of this package. This API is intended to be used by the package client (ideally, **IElement\_stream** only)

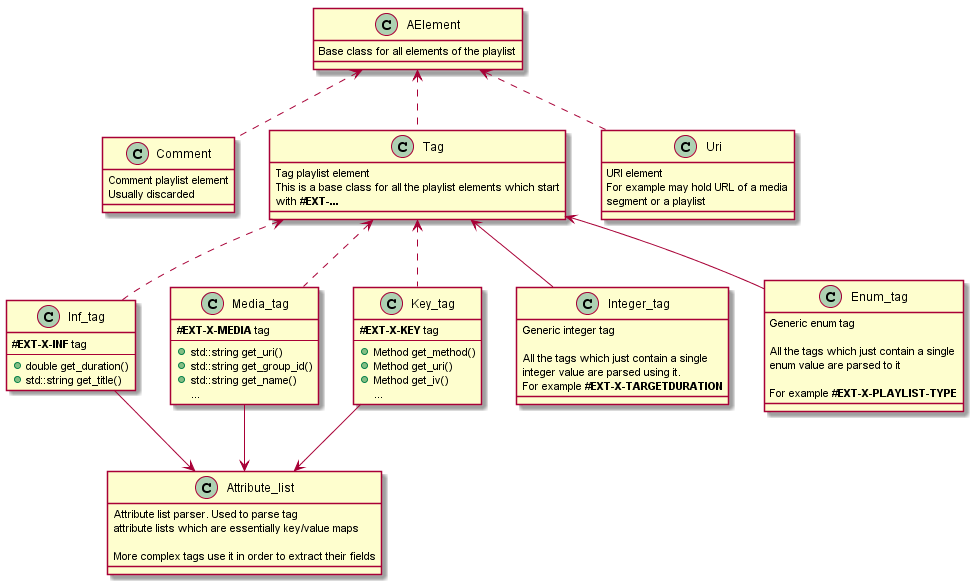


The figure above covers the basics of:

1. Pumping data into the parser. It’s done via the **ILine\_reader** interface, and its default implementation **String\_line\_reader**.  
   Of course this is a dependency injection, meaning the source itself can be anything the client defines.
2. Parsing actual elements (lines) of the playlist. This is done through the **IParser** interface and its default implementation **Parser**This is another dependency injection point, meaning the parser implementation may easily be swapped out (or mocked)
3. Providing a continuous “stream” of playlist elements using a **IElement\_stream** and its default implementation **Parser\_element\_stream**

### Data parsing

Figure below illustrates the internal model of the playlist elements. Essentially all playlist tags are defined by a single class, while some tags even share the same class (e.g. **Integer\_tag**)



#### AElement

We begin with the **AElement** abstract class. This is the base class for all the elements of the playlist, and as such all the elements inherit it.

Each element type is identified by an enumeration accessible through the base class:

* AElement::Type::**tag** - Identifies a **Tag** class element
* AElement::Type::**comment** - Identifies a **Comment** class element
* AElement::Type::**uri** - Identifies an **Uri** class element

For more info see:

* <http://mcs65:8080/xref/libhls2/libhls/hls/include/hls/m3u8/AElement.h>

Tag

This is the base class for all the tags of the manifest (e.g. all the lines which start with **#EXT-X**)

Each type of the tag is identified by an enum (similarly to how each element type is identified)

For example:

* Tag::Tag\_type::**m3u8**
* Tag::Tag\_type::**x\_media**
* Tag::Tag\_type::**x\_session\_key**
* Tag::Tag\_type::**x\_bitrate**

Etc..

For more info see:

* <http://mcs65:8080/xref/libhls2/libhls/hls/include/hls/m3u8/Tag.h>

#### Attribute\_list

This is the workhorse of the parsing package. It parses the key-value map that is the HLS Attribute List (for more info see **rfc8216bis-06 / 4.2. Attribute Lists** )

All the complex tags which contain multiple values are using this class to obtain them

For more info see:

* <http://mcs65:8080/xref/libhls2/libhls/hls/include/hls/m3u8/Attribute_list.h>
* <http://mcs65:8080/xref/libhls2/libhls/hls/source/m3u8/Attribute_list.cpp>

#### Integer\_tag

This is a class for all the tags which are defined by a single integer value

For more info see:

* <http://mcs65:8080/xref/libhls2/libhls/hls/include/hls/m3u8/Integer_tag.h>

#### Enum\_tag

This is a templated generic class which holds a single enumeration type. All the tags which are defined as such are using it.

It works together with an enum parsing system, to provide a uniform way of parsing enum values. For an example of how the parsing is used see how **Key\_tag::Method** enum:

* Enum definition:  
  <http://mcs65:8080/xref/libhls2/libhls/hls/include/hls/m3u8/Key_tag.h?r=192471a2#30>
* String to enum parser mapping:  
  <http://mcs65:8080/xref/libhls2/libhls/hls/source/m3u8/Key_tag.cpp?r=192471a2#16>

For more info see:

* <http://mcs65:8080/xref/libhls2/libhls/hls/include/hls/m3u8/Enum_tag.h>

#### Source code

Source code of the package can found here:

* Headers: <http://mcs65:8080/xref/libhls2/libhls/hls/include/hls/m3u8/>
* Source: <http://mcs65:8080/xref/libhls2/libhls/hls/source/m3u8/>
* Tests:
  + <http://mcs65:8080/xref/libhls2/libhls/test/source/TestParser.cpp>
  + <http://mcs65:8080/xref/libhls2/libhls/test/source/TestTags.cpp>
  + <http://mcs65:8080/xref/libhls2/libhls/test/source/TestVariableResolver.cpp>
  + <http://mcs65:8080/xref/libhls2/libhls/test/source/TestStreamInfTag.cpp>
  + [http://mcs65:8080/xref/libhls2/libhls/test/source/TestAttributeList.cpp](http://mcs65:8080/xref/libhls2/libhls/test/source/TestAttributeList.cpp?r=192471a2)
  + <http://mcs65:8080/xref/libhls2/libhls/test/source/TestKeyTag.cpp>
  + <http://mcs65:8080/xref/libhls2/libhls/test/source/TestMediaTag.cpp>

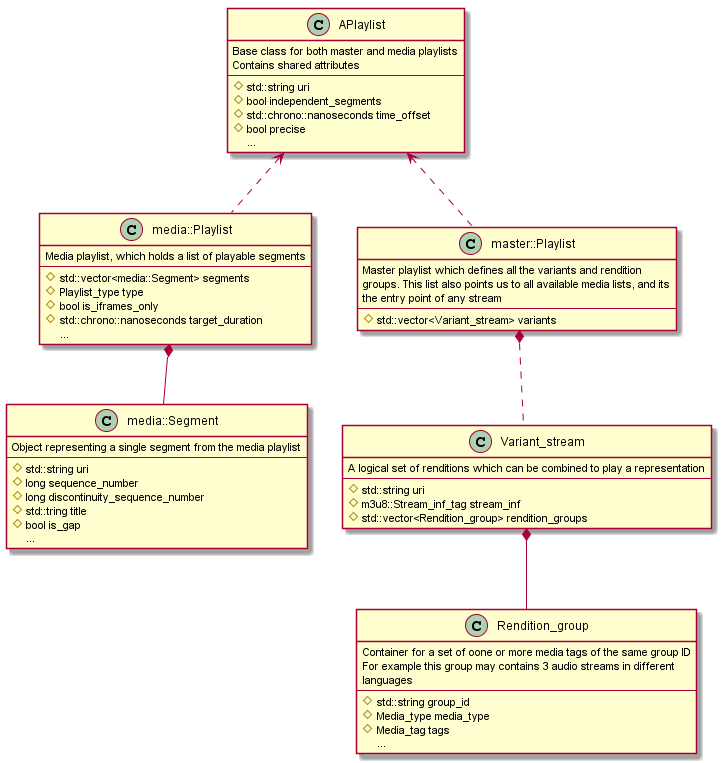
## Playlist package

This is the client of the m3u8 package. It provides the user with all the APIs needed to parse master and media playlists.

It works with elements exposed by the m3u8 package, and is not concerned by how the parsing is actually done, nor where the individual playlist elements are coming from.

The package is also not concerned on how the information exposed by it is used. It is just here to organize raw m3u8 elements into models defined by the HLS standard.

Figure below illustrates the top level view of the package



## 

#### APlaylist

Base class for both the master and media playlist. This class intended to store all the shared information between the two

#### media::Playlist

Media playlist is essentially a list of segments which can be downloaded. In addition to the segments, this playlist also contains various metadata (per segment or playlist-wide)

#### master::Playlist

Master playlist is the entry point of each HLS stream. It points to media playlists, and it defines stream-wide metadata.

Top level interface of the playlist revolves around the **Variant\_stream** concept. Each variant stream is essentially represented by a single **Stream\_inf** tag. This tag in turn points to a media playlist which contains all, or part of the media data

Each variant stream may or may not have rendition groups (e.g. alternative audio, etc.) associated with it

The master playlist deals with organizing all data from the master playlist into what is essentially a list of **Variant\_streams**.

Each **Variant\_stream** then points the library user to specific media playlists, etc.

#### Source code

Source code of the package can be found here:

* Headers

<http://mcs65:8080/xref/libhls2/libhls/hls/include/hls/playlist/>

* Source files:  
  <http://mcs65:8080/xref/libhls2/libhls/hls/source/playlist/>
* Tests
  + <http://mcs65:8080/xref/libhls2/libhls/test/source/TestMasterPlaylist.cpp>
  + <http://mcs65:8080/xref/libhls2/libhls/test/source/TestMediaPlaylist.cpp>

## Core package

Core package is as of right now not yet designed/developed.

This chapter only lists the responsibilities that it should hold:

* Client of the **Playlist** package
* Implements Segment streams (similar to libdash / **Adaptation\_stream**)
* Adds segment download functionality
* Adds support for ABR
* Adds top level entry point for libhls clients
* Adds top level stream management API (open/close tracks/switch representations etc.)