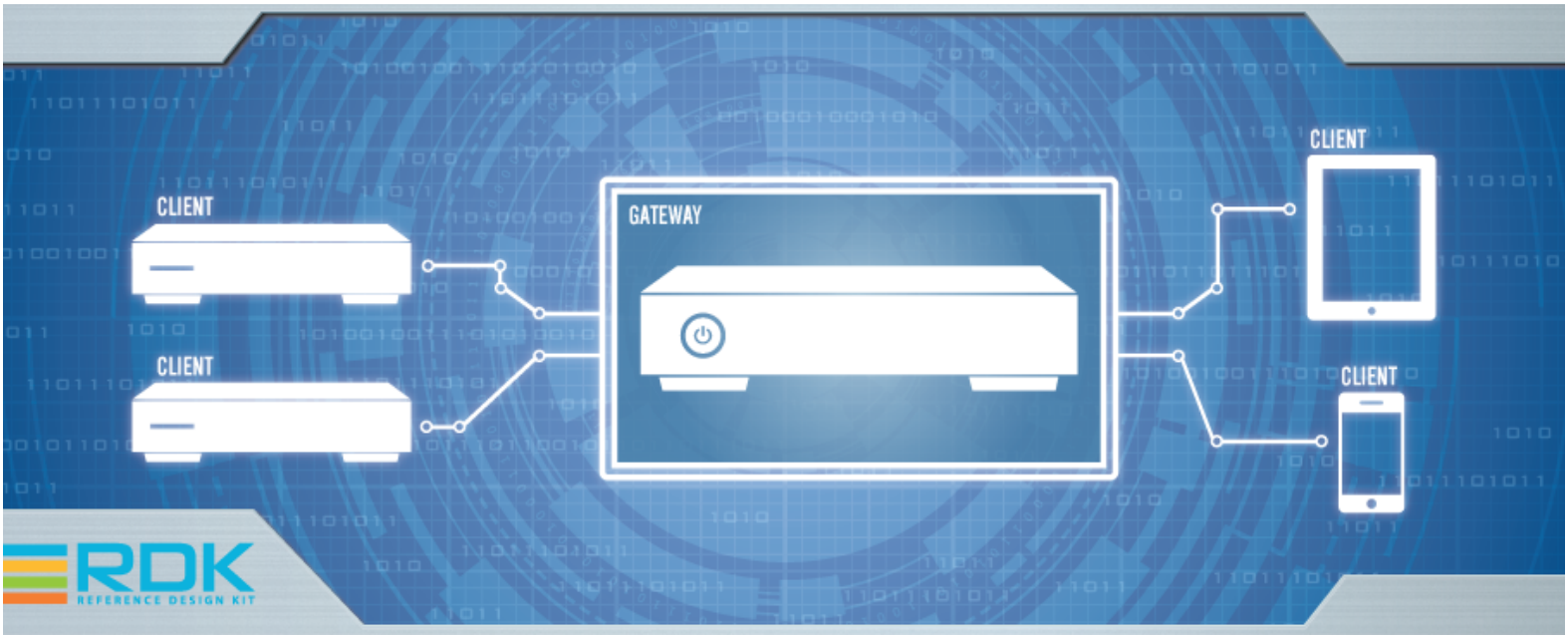

RDK
REFERENCE DESIGN KIT

RDK GStreamer Support Guidelines

TABLE OF CONTENTS

1 Introduction

RDK Media Framework (RMF) uses gstreamer as a playback mechanism for various types of media available on CPE devices. RMF acts as primary application for managing gstreamer based playback. RMF, with help of gstreamer can be used to play linear QAM video, Home Network Content, HTTP Dynamic/Live/Smooth Streaming and other media streams.

1.1 About This Document

This document describes the guidelines for gstreamer plugins on CPE devices which support RDK Media Framework.

1.1.1 Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1. MUST – This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
2. MUST NOT – This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
3. SHOULD – This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
4. SHOULD NOT – This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

1.1.2 Terminology

This document uses the following acronyms:

RDK	Settop box Reference Development Kit
HTTP	Hypertext Transfer Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
CPE	Consumer Premises Equipment

1.2 Intended Audience

This document is intended for engineers interested in the technical aspects of the RDK GStreamer requirements and guidelines.

1.3 Revision History

Version	Release Date	Description
0.1	9/19/11	Initial draft for internal review
0.2	9/21/11	Added DTCP/IP support and captioning sink properties
0.3	9/25/11	Implemented observations from Vinod and Fluendo
0.4	10/25/11	Updated with name and description of the gstreamer elements and properties Updated with information from Mark Rollins regarding Smooth Player
0.5	10/27/11	Added observations from Mark Rollins 1) Audio rate and channel capabilities 2) Capsfilter element Added dtcpip filter element

1.4 Reference Documents

Document Name	Revision	Doc Number
GStreamer documentation - http://gstreamer.freedesktop.org/documentation/		
Fluendo GStreamer elements – https://core.fluendo.com/gstreamer/trac/wiki		

2 GStreamer Overview

GStreamer is a framework for creating streaming media applications. GStreamer's development framework makes it possible to write any type of streaming multimedia application. GStreamer framework is designed to make it easy to write applications that handle audio or video or both. It isn't restricted to audio and video, and can process any kind of data flow. The pipeline design is made to have little overhead above what the applied filters induce. This makes GStreamer a good framework for designing even high-end audio applications which put high demands on latency.

The framework is based on plugins that will provide the various codec and other functionality. The plugins can be linked and arranged in a pipeline. This pipeline defines the flow of the data. Pipelines can also be edited with a GUI editor and saved as XML so that pipeline libraries can be made with a minimum of effort.

3 GStreamer Pipeline Requirements

3.1 Version

RDK Media Framework has been verified using gstreamer version 0.10.28. Any version better than 0.10.28 have to be backwards compatible with 0.10.28.

3.2 Supported media types

GStreamer framework along with elements on any platform must have the ability to playback the following media types.

- MPEG-2 (Transport Stream) Video with MPEG-1 Layer II (MP2) or AC3 audio
- MPEG-4/H.264 Video with AAC audio
- MP3 audio
- WAV (GSM)

3.3 GStreamer elements

All elements used in the gstreamer pipeline should adhere to the gstreamer framework specification. They must support all standard states, events and messages of gstreamer.

3.4 Hardware Support

Based on platform support, gstreamer elements should take support of hardware acceleration. All demux, decoder and sink related elements are recommended and encouraged to make use of Native platform hardware interface.

3.5 Debug/Test support

All platforms should support gstreamer related tools like gst-debug, gst-inspect and gst-launch. At a minimum they should provide a test application which provides the same functionality as gst-launch.

3.6 Build

Subject to licensing, gstreamer base framework and all gstreamer elements should be available in source from RDK svn and can be built using the RDK build infrastructure.

3.7 DTCP/IP support

GStreamer pipeline must support DTCP/IP decryption for mpeg2 ts streams. This support could be in form of a new element which does decryption before sending stream to the pipeline or additional property of one of the existing elements.

3.8 Performance

GStreamer pipeline should start playing/showing video/audio within 250ms after the first packet available at source.

3.9 Pause/Resume

Using the pipeline it must be possible to smoothly pause (freeze picture) and resume smoothly from the paused picture.

3.10 Dynamic source change

Using the pipeline it must be possible to freeze picture, change source (file/http) dynamically, and resume playback from the new source without glitches, and without requiring close/re-open of the pipeline.

4 GStreamer plugin requirements

To construct and control the gstreamer pipeline from the XRE Nativer Recevier, GStreamer elements used in the pipeline must support few properties.

4.1 Source elements

Based on the device type and capabilities RMF requires the support of several gstreamer source elements. Here are the major gstreamer source elements RMF would require

1) Http Source
2) File Source
3) QAM source
4) Fake Source

In most of the RDK versions, several source elements should be readily available.

For example: souphttpsrc, filesrc, fakesrc etc.

4.1.1 HTTP Source

All RDK versions must support HTTP Source. Http source elements must support the following properties.

1) 'location'
2) 'startPTS'
3) 'endPTS'
4) 'proxy' - Proxy server to use, in the form HOSTNAME:PORT
5) 'user-agent' - Value of the User-Agent HTTP request header field

In general Http Source must support these actions/methods.

1. Accept URL along with the query parameters
2. Set and Get HTTP header fields from HTTP Requests and Responses
3. Support Chunked encoding
4. Read Http Error codes
5. Support and Control buffer sizes
6. Set trick-mode property to instruct demux and other elements to go to trick-mode

4.1.2 QAM/DVB Source

Based on the RDK version and type, gstreamer source elements should provide the ability to support QAM based tuning.

QAM Source must support the following properties.

1) Frequency – Read/Write - Tune Frequency
2) Modulation – Read/Write – QAM modulation
3) retries – Read/Write – Max Tune retries
4) tunerid – Read
5) pgmno – Read/Write – Program Number
6) dmxhdl – Read – Demux Handle
7) pidarray – Write – Array of pids to be filtered, first item should be PMT pid

4.1.3 File Source

RMF utilizes File Source (filesrc) element, which is part of standard gstreamer core elements package

4.1.4 Fake Source

RMF utilizes Fake Source (fakesrc) element, which is part of standard gstreamer core elements package

4.2 Demux elements

Demux or similar elements must be able to link to source elements with or without queue/buffer elements.

RMF currently contains/make use of several demuxers based on use case scenario.

1) Mpeg2/ts demuxer - 'flutsdemux' - This is an open source soft demux from Fluendo modified for RMF trick modes support
2) Mpeg4/h264 demuxer - 'qtdemux' – Open Source h264 demux from gst-plugins-good
3) Mp3 Audio demuxer – 'id3demux' – Open Source demux from gst-plugins-good

4.2.1 Mpeg/ts demux

Mpeg2/TS Demux must support the following properties

1) 'pat-info' - information from the TS PAT about all programs listed in the current Program Association Table (PAT)
2) 'pmt-info' - information from the TS PMT about the currently selected program and its streams (which includes program-number, pcr-pid, stream-info etc.)

```
g_object_get (obj, "pmt-info", &pmtinfo, NULL);
  g_object_get (pmtinfo, "program-number", &program, NULL);
  g_object_get (pmtinfo, "version-number", &version, NULL);
  g_object_get (pmtinfo, "pcr-pid", &pcr_pid, NULL);
  g_object_get (pmtinfo, "stream-info", &streaminfos, NULL);
  g_object_get (pmtinfo, "descriptors", &descriptors, NULL);

  g_print ("PMT: program: %04x version: %d pcr: %04x streams: %d "
      "descriptors: %d\n",
      (guint16)program, version, (guint16)pcr_pid, streaminfos->n_values,
      descriptors->n_values);

  dump_descriptors (descriptors);
  for (i = 0 ; i < streaminfos->n_values; i++) {
    value = g_value_array_get_nth (streaminfos, i);
    streaminfo = (GObject*) g_value_get_object (value);
    g_object_get (streaminfo, "pid", &es_pid, NULL);
    g_object_get (streaminfo, "stream-type", &es_type, NULL);
    g_object_get (streaminfo, "languages", &languages, NULL);
    g_object_get (streaminfo, "descriptors", &descriptors, NULL);
    g_print ("pid: %04x type: %x languages: %d descriptors: %d\n",
        (guint16)es_pid, (guint8) es_type, languages->n_values,
       descriptors->n_values);
    dump_languages (languages);
    dump_descriptors (descriptors);
  }
```

In general, mpeg2/ts demuxer must support these actions/methods

1) Dynamically linking/delinking Audio/Video pad's (based on pmt-info)
2) Switch to trick-play mode and turn off the packet continuity inspection

4.3 Decoder elements

Decoder or similar elements must be able to link to the demux elements with or without queue/buffer elements.

RMF needs support for decoding the following media types

1) Decoder for Mpeg2/TS media
2) Decoder for Mpeg4/h264 media

Mpeg2 TS decoder must support the following properties

1. 'trick-rate' - trick-play client support
2. 'interlaced' – interlaced property of the stream
3. 'decode-handle' – Decoder identifier which can be used to retrieve user data from decoder

Mpeg2 Decoder must support following functionality.

1. Smooth forward decode rates: For smooth forward modes (0 < speed <= 2), decoder should be able to decode all frames at the set rate

2. Smooth GOP reverse mode : For smooth rewind (speeds <0 and > -2) Media Streamer will send the GOP's in reverse order, hence the decoder must have the capability to decode the frames in each GOP in reverse order.
4. Frame mask – Decoder must be able to decode and present selected frame types (I-Frame/P-Frame/B-frame or any combination).
5. Should allow getting the interlaced property of the stream. This is useful while switching the video sink to fixed frame rate mode.

4.4 A/V Sink elements

RMF Player needs atleast one videosink and audiosink gstreamer plugins for rendering audio/video.

1) Video Sink
2) Audio Sink

'vidsink' must support the following properties

1) 'tvmode' - Define the television mode
2) 'plane' - Define the Pixel Plane to be used
3) 'rectangle' - The destination rectangle
4) 'flush-repeat-frame' - Keep displaying the last frame rather than a black one whilst flushing
5) 'currentPTS'
6) 'inter-frame-delay' - Enables fixed frame rate mode
7) 'slow-mode-rate' - slow mode rate in normalised form(-20000 < 0 <=20000)
8) 'contentframerate' - get content frame rate
9) 'stepframe'
10) 'mute'

Vidsink must support the following actions/functionality on the pipeline:

1. Scaling/Rectangle – Video sink should support scaling by using at least 4 co-ordinates (x, y, height, width)
2. Transition between the various scaling modes must be smooth
3. Fixed frame rate mode – Video Renderer/Sink must be able to set fixed frame rate property. The video should then be presented at the set frame rate (e.g. 10 fps) instead of the frame rate specified in the stream. The rate can be set as an inter-frame-delay in PTS ticks. For e.g. 9000 would mean 10 frames per sec.
4. Frame step - Step forward/backward one frame at a time. In case of backward frame step, stream will be provided GOP reversed.
5. Get the last presented PTS value: This is required to generate the media time
6. Plane selection – Video sink should have an option to select which plane the video output should render in case of more than one video plane.

'audio_sink' must support the following properties

1) 'volume' – volume factor
2) 'mute' – mute a channel
3) 'audio-output-hdmi' – Define the audio output hdmi mode
4) 'audio-output-spdif' – Define the audio spdif mode
5) 'audio-output-<any other>'

'audio_sink' must also support following capabilities:

1) audio/x-raw-int
2) audio/mpeg
3) audio/x-aac
4) audio/x-ac3
5) audio/x-gsm
6) rate
7) channels
8) Any other types defined in the Hardware specifications

4.5 Parse elements

RMF would require the following parse elements for playing different audio formats.

1) For mp3 audio – 'mp3parse'
2) For gsm/wav audio – 'wavparse'

Properties required for Mp3parse or other related elements

1) Seek – Should be able to seek any position in a mp3 audio stream

4.6 Filter elements

RMF needs the following filter elements

1) 'capsfilter' from gst-core-elements
2) 'dtcpip' to decrypt the DTCP/IP content

*dtcpip decryption can also be part of other source elements in the media pipeline.

4.7 Closed Captioning

RMF would require a property on the decoder element to give the decoder index out to the UI application. UI can initialize CC Reader to get the user data from decoder index and pass it on to CC Manager.

1) RMF needs "decoder-handle" property which points to decoder index being used on the current media pipeline.

5 Appendix A – Sample pipeline for various media streams

mpeg2 ts streams:

gst-launch souphttpsrc location="http://1.2.3.4:8080/vldms/dvr?rec_id=1316099428081" ! flutsdemux name=d ! queue ! mpeg2_viddec ! vidpproc ! vidrend_sink d. ! queue ! audio_sink audio-output1=0

mpeg4 h264 streams:

gst-launch-0.10 filesrc location=sample.mp4 ! qtdemux name=demuxer demuxer. ! queue ! h264_viddec ! queue ! vidpproc ! vidrend_sink demuxer. ! queue ! audio_sink

mp3 audio streams:

gst-launch-0.10 filesrc location=voicemail.mp3 ! id3demux ! mp3parse ! audio_sink

6 Appendix B – MS Smooth Playback mechanism using gstreamer pipeline

This section explains the MS Smooth Playback mechanism with/without using Playready DRM.

- Smooth Player reads the manifests and streams video from Server.
- Smooth Player acts as demux by reading and processing audio and video streams independently.
- If the streams are encrypted, they will be passed to Manufacturer Playready DRM module for decryption.
- Smooth Player uses two independent gstreamer pipelines for audio and video playback. Hence it does not make use of demux element.
- It directly pushes the streams into decoder buffers using ‘fakesrc’ element and gstreamer callbacks for more data.