Advanced Micro Devices

# Advanced Media Framework – Display Capture

Programming Guide

## Disclaimer

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information.

Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, ATI Radeon™, CrossFireX™, LiquidVR™, TrueAudio™ and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Windows™, Visual Studio and DirectX are trademark of Microsoft Corp.

## Copyright Notice



Notice Regarding Standards. AMD does not provide a license or sublicense to any Intellectual Property Rights relating to any standards, including but not limited to any audio and/or video codec technologies such as MPEG-2, MPEG-4; AVC/H.264; HEVC/H.265; AAC decode/FFMPEG; AAC encode/FFMPEG; VC-1; and MP3 (collectively, the "Media Technologies"). For clarity, you will pay any royalties due for such third party technologies, which may include the Media Technologies that are owed as a result of AMD providing the Software to you.

## MIT license

## Contents



# 1 Introduction

The AMD Advanced Media Framework (AMF) includes functionality for display capture to facilitate various streaming solutions in remote display, network game streaming and other applications. The display capture function is designed to work in conjunction with other AMF components, such as H.264, H.265 and AV1 encoders and color space converter.

AMF currently offers two components to perform display capture. One legacy component is using the Microsoft DXGI Desktop Duplication API (DD), while the new component utilizes an AMD's proprietary API to perform screen capture. The new display capture API is not available in legacy drivers, however it provides a more efficient way to perform display capture with lower latency and lower impact on the CPU and GPU performance. It is therefore recommended to use the new API for solutions where compatibility with legacy drivers is not required.

Functionally both methods are equivalent and implement the same API. The legacy DD component is available in the source code form as a sample.

Note: The Display Capture API requires root or super user privileges when running on Linux systems.

# 2 Display Capture Programming Model

AMF provides a standard component implementing the `AMFComponent` interface to perform display capture. For more information about the `AMFComponent` interface please refer to Section 2.6.1 of the AMF API Reference.

The Display Capture component is a source and does not take any input.

## 2.1 Creating the Display Capture Component

To create an instance of the Display Capture component, call the `AMFFactory::CreateComponent()` method passing `AMFDisplayCapture` as parameter. Include the `public/include/components/DisplayCapture.h` header.

The open source legacy Display Capture component based on the Microsoft DXGI Desktop Duplication API is included in the AMF samples in form of source code. It can be created by calling the `AMFCreateComponentDisplayCapture` function defined in `public/src/components/DisplayCapture/DisplayCaptureImpl.cpp`. Refer to the public DVR sample for details.

## 2.2 Initializing the Display Capture Component

The Display Capture component is initialized by calling the `AMFComponent::Init` method. Prior to calling `AMFComponent::Init` a number of properties must be set on the component object using the `AMFPropertyStorage::SetProperty` method:

| Name | Type |
|---|---|
| AMF_DISPLAYCAPTURE_MONITOR_INDEX | amf_int64 |

| Name | Type |
| --- | --- |
| AMF_DISPLAYCAPTURE_FRAMERATE | AMFRate |

Table 1. Properties of the SetProperty method

---

**Name:** `AMF_DISPLAYCAPTURE_MONITOR_INDEX`

**Values:** `IDXGIFactory::EnumAdapters` for Windows, index on `libdrm` display enumeration on Linux

**Default Value:** `0`

**Description:** A monitor index to capture, determined by calling `IDXGIFactory::EnumAdapters`, 0 specifies the default monitor.

---

**Name:** `AMF_DISPLAYCAPTURE_FRAMERATE`

**Values:** Desired capture output framerate for `FRAMERATE` mode

**Default Value:** `(0,1)`

**Description:** Frame rate to perform the capture at. Setting the numerator to 0 causes the capture to be performed at the rate defined by either the application's flip frequency (for full-screen applications) or by DWM (for windows applications).

---

You can implement custom control of timestamps on each captured frame by providing a custom implementation of the `AMFCurrentTime` interface defined in `public/include/core/CurrentTime.h` and assigning it to the `AMF_DISPLAYCAPTURE_CURRENT_TIME_INTERFACE` property. By default, when the `AMF_DISPLAYCAPTURE_CURRENT_TIME_INTERFACE` property is not set, timestamps are assigned the value returned by `amf_high_precision_clock()` function at the time when a frame is captured.

Once the properties are set, call the `AMFComponent::Init` method. Pass `AMF_SURFACE_UNKNOWN` for `format` and zeros for `width` and `height`.

Once successfully initialized, the Display Capture component can be queried for output.

Upon initialization, the following properties can be read using the `AMFPropertyStorage::GetProperty` method:

| Name | Type |
| --- | --- |
| AMF_DISPLAYCAPTURE_FORMAT | amf_int64 |
| AMF_DISPLAYCAPTURE_RESOLUTION | AMFSize |
| AMF_DISPLAYCAPTURE_ROTATION | AMF_ROTATION_ENUM |

Table 2. Properties of GetProperty method

---

**Name:** `AMF_DISPLAYCAPTURE_FORMAT`

**Values:** `AMF_SURFACE_FORMAT`

**Default Value:** `N/A`

**Description:** Capture format ( `AMF_SURFACE_FORMAT`).

---

**Name:** `AMF_DISPLAYCAPTURE_RESOLUTION`

**Values:** A valid size.

**Default Value:** `N/A` or `(0,0)`

**Description:** Captured image resolution; An output parameter representing actual screen/display size.

---

**Name:** `AMF_DISPLAYCAPTURE_ROTATION`

**Values:** `AMF_ROTATION_ENUM` : `AMF_ROTATION_NONE`, `AMF_ROTATION_90`, `AMF_ROTATION_180`, `AMF_ROTATION_270`

**Default Value:** `AMF_ROTATION_NONE`

**Description:** Rotation of monitor being captured, `AMF_ROTATION_NONE` by default.

---

## 2.3 Querying for Output

The output of the Display Capture component can be obtained by calling the `AMFComponent::QueryOutput` method in a loop. The loop needs to run fast enough to sustain the frame rate set during initialization using the `AMF_DISPLAYCAPTURE_FRAMERATE` property. When a frame is available, `AMFComponent::QueryOutput` places a pointer to the `AMFSurface` object at the location pointed to by the `ppData` parameter. When no new frame is available yet, `ppData` is set to `NULL` and `AMFComponent::QueryOutput` returns `AMF_REPEAT`.

As with any other AMF component, it is recommended to run the polling loop in a separate thread. Whenever `AMFComponent::QueryOutput` returns `AMF_REPEAT`, the polling thread should be put to sleep for at least 1 ms to avoid high CPU utilization.

The `AMFSurface` object containing a captured frame that was obtained from `AMFComponent::QueryOutput` can be used as input for the next component in the pipeline.

The capture contained in the returned `AMFSurface` can also be modified with the following properties:

`AMF_DISPLAYCAPTURE_DUPLICATEOUTPUT`, of type `amf_bool`, false by default. If set, the frame returned in the `AMFSurface` object will be a copy the last captured output.

`AMF_DISPLAYCAPTURE_ENABLE_DIRTY_RECTS`, of type `amf_bool`, false by default. If set, dirty rectangles indicating changed areas in frame since last output are attached to the returned `AMFSurface` as the property `AMF_DISPLAYCAPTURE_DIRTY_RECTS` (See Section 2.5).

`AMF_DISPLAYCAPTURE_DRAW_DIRTY_RECTS`, of type `amf_bool`, false by default. If set, the captured output in the `AMFSurface` will have the dirty rectangles drawn in red. For debugging purposes only.

## 2.4 Shutting Down Display Capture

To stop display capture, call `AMFComponent::Drain`. You can exit the polling loop and terminate the polling thread once `AMFComponent::QueryOutput` returns `AMF_EOF`.

Call `AMFComponent::Terminate` and release the pointer to the Display Capture component.

## 2.5 Capture modes

Application can select three capture modes by setting `AMF_DISPLAYCAPTURE_MODE` into one of three modes:

| Name | Type |
|---|---|
| AMF_DISPLAYCAPTURE_MODE_KEEP_FRAMERATE | AMF_DISPLAYCAPTURE_MODE_ENUM |

| Name | Type |
| --- | --- |
| AMF_DISPLAYCAPTURE_MODE_WAIT_FOR_PRESENT | AMF_DISPLAYCAPTURE_MODE_ENUM |
| AMF_DISPLAYCAPTURE_MODE_GET_CURRENT_SURFACE | AMF_DISPLAYCAPTURE_MODE_ENUM |

Table 3. AMF Capture modes

---

**Name:** `AMF_DISPLAYCAPTURE_MODE_KEEP_FRAMERATE`

**Value:** `0`

**Description:** Component will keep requested framerate, repeating frame if new present didn't happen

---

**Name:** `AMF_DISPLAYCAPTURE_MODE_WAIT_FOR_PRESENT`

**Value:** `1`

**Description:** Component returns captured frame with presentation rate: DWM or full screen app.

---

**Name:** `AMF_DISPLAYCAPTURE_MODE_GET_CURRENT_SURFACE`

**Value:** `2`

**Description:** Component returns current frame immediately.

---

If available, the output surface will have the following properties:

| Name | Type |
| --- | --- |
| AMF_DISPLAYCAPTURE_FRAME_INDEX | amf_int64 |
| AMF_DISPLAYCAPTURE_FRAME_FLIP_TIMESTAMP | amf_int64 |
| AMF_DISPLAYCAPTURE_DIRTY_RECTS | AMFBufferPtr |

Table 4. Output surface properties

---

**Name:** `AMF_DISPLAYCAPTURE_FRAME_INDEX`

**Values:** <= `0`

**Default Value:** `0`

**Description:** Index of present call for the current captured frame starting from beginning of capture.

---

**Name:** `AMF_DISPLAYCAPTURE_FRAME_FLIP_TIMESTAMP`

**Values:** <= `0`

**Default Value:** `0`

**Description:** Flip timestamp of the presented frame acquired by `QueryPerformanceCounter()`.

---

**Name:** `AMF_DISPLAYCAPTURE_DIRTY_RECTS`

**Values:** `AMFBufferPtr`

**Default Value:** `N/A`

**Description:** Array of `AMFRect` objects indicating changed areas on the captured surface since the last capture. The structure of `AMFRect` can be found in `public/include/core/Platform.h`.

---