Intel® Media Server Studio 2017 Video Quality Caliper Reference Manual

for Metrics Plugin API

API Version 1.0

LEGAL DISCLAIMER

Table of Contents

Overview

The **Intel® Media Server Studio 2017 - Video Quality Caliper** supports the implementation of custom metrics via a C++ plugin interface (API). Custom metrics plugins are implemented as shared libraries (*.dll or *.so) and this document describes the functions and data structures which allow users to create custom metrics. Sample code to build a typical plugin is also provided in the package vqcaliper_plugin_samples.zip. The sample code may be used as a template for creating additional custom metrics.

Document Conventions

This reference manual uses the Verdana typeface for normal prose. With the exception of section headings and the table of contents, all code-related items appear in the `Courier New` typeface (`vqcStatus` and `Init`). All class-related items appear in all cap boldface, such as **DECODE** and **ENCODE**. Member functions appear in initial cap boldface, such as **Init** and **Reset**, and these refer to members of all classes, **DECODE**, **ENCODE**. Hyperlinks appear in underlined boldface, such as **vqcStatus**.

Acronyms and Abbreviations

VQC	Video Quality Caliper
API	Application Programming Interface

Version History

Intel® Media Server Studio 2017 – Video Quality Caliper – Metrics Plugin API:

Version 1.0

- Initial implementation

System Requirements

The VQC Metrics Plugin API is supported on the same platforms as the **Intel® Media Server Studio 2017 - Video Quality Caliper** with which it was packaged. See the manual Video_Quality_Caliper_Release_Notes.pdf for the system requirements corresponding to your installation.

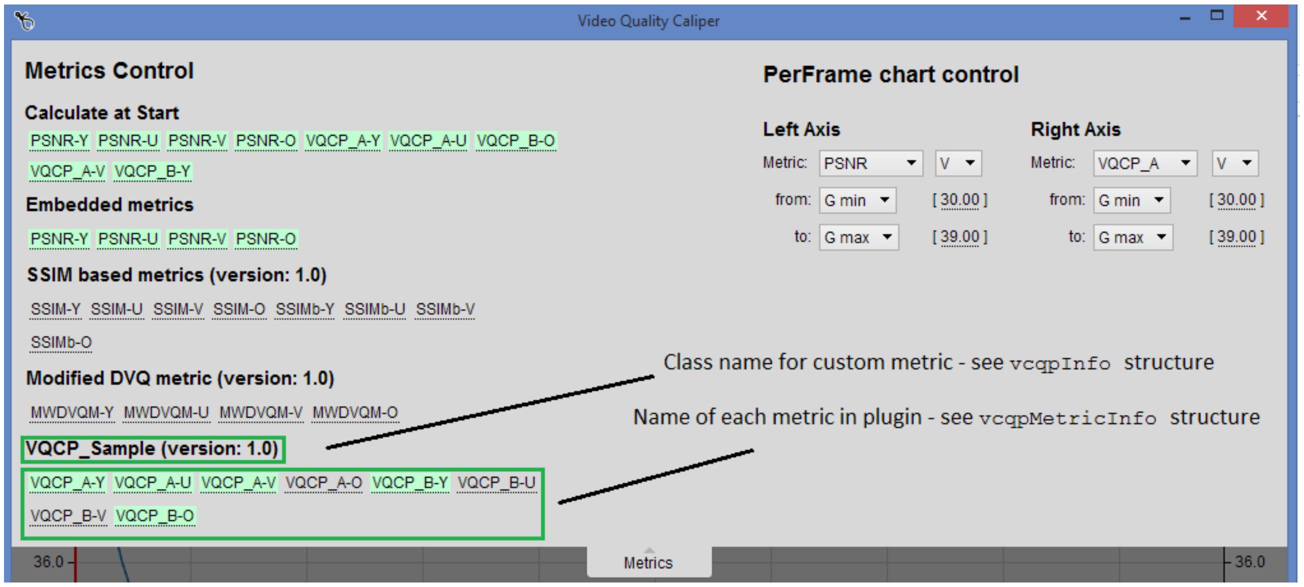
Building a custom metrics plugin also requires a C++ toolchain which can create shared libraries for your platform (*.dll on Windows, *.so on Linux). The sample code includes project files for Microsoft Visual Studio 2013 (Windows) and the GNU toolchain (Linux), but other toolchains may also be used if desired.

Installation

The VQC Metrics Plugin API provides C++ header files (*.h) which define the functions and datatypes necessary to implement a custom plugin. Sample code to build an example plugin is also provided. The header files, sample code, and build instructions are included in the package vqcaliper_plugin_samples.zip which is installed in the same location as the VQC application executable. It is recommended to extract the contents of the sample code package to a convenient location. The sample code package includes a readme.txt with instructions for building the example plugin.

Architecture

The VQC Metrics Plugin API allows users to implement custom metrics for use with the Video Quality Caliper application. Custom metrics plugins are built as shared libraries (*.dll or *.so) implementing the C++ interface described in this document. The VQC application searches the `plugins\` subdirectory within the main VQC installation directory for any custom metrics plugins implemented by the user. The custom metrics can be selected and configured in the Metrics page of the VQC application, in the same manner as the metrics provided with VQC, as shown below:


Video Quality Caliper
Metrics Control
Calculate at Start
PSNR-Y PSNR-U PSNR-V PSNR-O VQCP_A-Y VQCP_A-U VQCP_B-O
VQCP_A-V VQCP_B-Y
Embedded metrics
PSNR-Y PSNR-U PSNR-V PSNR-O
SSIM based metrics (version: 1.0)
SSIM-Y SSIM-U SSIM-V SSIM-O SSIMb-Y SSIMb-U SSIMb-V
SSIMb-O
Modified DVQ metric (version: 1.0)
MWDVQM-Y MWDVQM-U MWDVQM-V MWDVQM-O
VQCP_Sample (version: 1.0)
VQCP_A-Y VQCP_A-U VQCP_A-V VQCP_A-O VQCP_B-Y VQCP_B-U
VQCP_B-V VQCP_B-O
PerFrame chart control
Left Axis
Right Axis
Metrics
36.0
36.0

Left Axis — Metric: PSNR, V; from: G min [30.00]; to: G max [39.00]

Right Axis — Metric: VQCP_A, V; from: G min [30.00]; to: G max [39.00]

Class name for custom metric - see `vcqpInfo` structure

Name of each metric in plugin - see `vcqpMetricInfo` structure

Development of the VQC Metrics Plugin API is ongoing and new functionality may be added in future versions. The following are some important known limitations of the current version:

- In each call to `CalculateFrame()`, the VQC application only provides the current frame from each reference and test stream. Past or future frames are not available. The fields `maxFramesPrev` and `maxFramesFutr` must be set to zero in `vqcpMetricInfo`.

- In each call to `CalculateFrame()`, the metrics plugin may only return one result per metric. The field `maxNumResults` must be set to 1 in `vqcpMetricInfo`, and the field `numResults` must be set to 1 in `vqcpResults`. Additional results will be ignored by the VQC application.

- During plugin initialization, the VQC application instructs the plugin which of the available metrics and channels to calculate, via a space-delimted list of metric names in `vqcpParams.subscribedMetrics` which enables the plugin to skip processing for metric/channel combinations which are not needed. The application may create more than one instance of each metrics plugin, with a different subset of metrics enabled in each instance. For example, selecting a metric/channel combination under “Per-Frame chart control” which was not initially selected as a “Calculate at Start” item will initialize a new instance of the `VQCMetricPlugin` object with the requested combination enabled. Therefore each custom metric plugin must be fully re-entrant and not share modifiable (non-const) data between different instances of the same object.

- The value of `vqcpParams.complexityScale` is not used in the current version, and should be set to zero.

Programming Guide

This chapter describes the concepts used in programming with the VQC Metrics Plugin API. Please refer to the relevant chapters in this document, as well as the header file `vqcplugins.h`, for additional information about specific functions and datatypes.

Custom metrics plugins are implemented as shared libraries (*.dll or *.so) which are loaded during startup of the VQC application. Each custom metrics plugin exports the following functions, which should be declared as `extern "C"` (i.e. without name-mangling).

```
vqcStatus   VQCP_QueryPlugin(vqcpInfo *pluginInfo)
vqcStatus   VQCP_CreateMetricInstance(C_VQCMetricPlugin **metricPlugin)
vqcStatus   VQCP_DestroyMetricInstance(C_VQCMetricPlugin *metricPlugin)
```

The VQC application searches the `plugins` subdirectory (relative to its installation path) and loads any shared library which properly implements this interface. Files which do not export these symbols are skipped. (Note: the VQC application also includes pre-built metrics plugins in the same directory, but these employ a different interface and may not be loaded via the process outlined in this document).

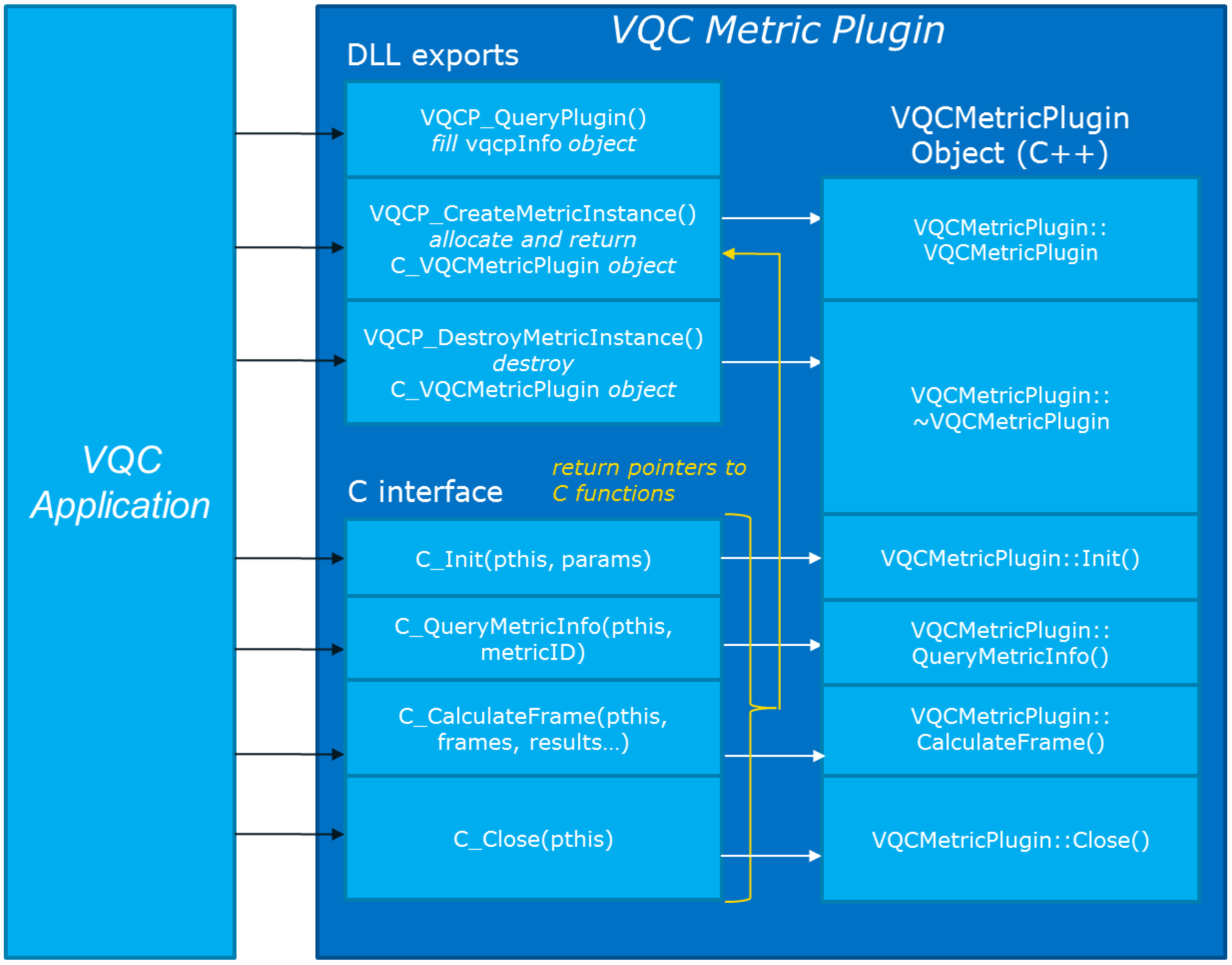
The function `VQCP_QueryPlugin` queries the plugin's capabilities (i.e. what metrics are supported, what names to display in the GUI, and other parameters describing each metric). The parameter `pluginInfo` is a pointer to a struct of type `vqcpInfo`. The plugin should fill in the fields of this structure with the appropriate values. Strings returned via `vqcpInfo` should be declared as `const` and not modified at any time while the shared library is loaded.

The function `VQCP_CreateMetricInstance` creates a new instance of the custom metric. For each instance, the plugin should allocate a structure of type `C_VQCMetricPlugin` and create a new instance of the plugin object (that is, the derived class which implements the pure virtual functions defined in `VQCMetricPlugin`) The plugin should then fill each member `C_VQCMetricPlugin` with the appropriate pointer, and return a pointer to this structure.

The function `VQCP_DestroyMetricInstance` destroys the instance of the custom metric which was allocated in `VQCP_CreateMetricInstance`. The plugin should first destroy the instance of the `VQCMetricPlugin` object (i.e. `C_VQCMetricPlugin->pthis`) and then delete the `C_VQCMetricPlugin` object itself.

To implement a custom plugin, the programmer defines a new class that inherits `VQCMetricPlugin` and implements its pure-virtual functions. The derived class may also define any number of private, implementation-specific functions and data structures, but these will not be visible to or used by the VQC application.

The C++ object created with each instance of the plugin is not used directly by the VQC application. Rather, the `C_VQCMetricPlugin` receives function pointers to thin C wrapper functions in the plugin, which in turn make the appropriate function calls into the C++ class. This is done to simplify the process of calling functions across the DLL boundary. The following diagram illustrates this architecture.


VQC Application
VQC Metric Plugin
DLL exports
VQCP_QueryPlugin() *fill* vqcpInfo *object*
VQCP_CreateMetricInstance() *allocate and return* C_VQCMetricPlugin *object*
VQCP_DestroyMetricInstance() *destroy* C_VQCMetricPlugin *object*
C interface
return pointers to C functions
C_Init(pthis, params)
C_QueryMetricInfo(pthis, metricID)
C_CalculateFrame(pthis, frames, results…)
C_Close(pthis)
VQCMetricPlugin Object (C++)
VQCMetricPlugin::VQCMetricPlugin
VQCMetricPlugin::~VQCMetricPlugin
VQCMetricPlugin::Init()
VQCMetricPlugin::QueryMetricInfo()
VQCMetricPlugin::CalculateFrame()
VQCMetricPlugin::Close()

Function Reference

This section describes VQC Metrics Plugin API functions and their operations. All of the functions listed in this section must be implemented for each custom metrics plugin.

In each function description, only commonly used status codes are documented. The function may return additional status codes, such as **`VQC_ERR_INVALID_HANDLE`** or **`VQC_ERR_NULL_PTR`**, in certain cases. See the **`vqcStatus`** enumerator for a list of all status codes.

Global Functions

Global functions perform query functions on a global scale and create/destroy new instances of the custom metric.

VQCP_QueryPlugin

Syntax

```
vqcStatus VQCP_QueryPlugin(vqcpInfo *pluginInfo);
```

Parameters

`pluginInfo`	Pointer to `vqcpInfo` structure

Description

The application calls this function at runtime to query the capabilities of each metrics plugin. The plugin should fill in each of the fields in `pluginInfo` with appopriate data describing the plugin. Strings should be declared as `const` and not modified while the plugin library is loaded into memory.

This function is exported in the shared library (DLL).

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

VQCP_CreateMetricInstance

Syntax

```
vqcStatus VQCP_CreateMetricInstance(C_VQCMetricPlugin **metricPlugin);
```

Parameters

`metricPlugin`	Pointer to return `C_VQCMetricPlugin` object

Description

The application calls this function to create a new instance of the metrics calculation object. The plugin creates a new object of type `C_VQCMetricPlugin`, and also a new instance of the user's implementation of the metric, i.e. a class derived from `VQCMetricPlugin`. The pointer to the newly-created `VQCMetricPlugin` object is stored in `C_VQCMetricPlugin->pthis`. The remaining members of `C_VQCMetricPlugin` are initialized with pointers to the corresponding C interface functions (`C_Init`, etc.)

The application will pass this `VQCMetricPlugin` object as a parameter when calling any C interface function. The plugin may then cast `pthis` to the class he implemented and call the corresponding `VQCMetricPlugin::` function.

This function is exported in the shared library (DLL).

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

VQCP_DestroyMetricInstance

Syntax

```
vqcStatus VQCP_DestroyMetricInstance(C_VQCMetricPlugin *metricPlugin);
```

Parameters

`metricPlugin`	Pointer to return `C_VQCMetricPlugin` object

Description

The application calls this function to destroy an instance of the metrics calculation object which was allocated by `VQCP_CreateMetricInstance`. The plugin should first call the destructor for the `VQCMetricPlugin` object represented by `C_VQCMetricPlugin->pthis`, and then destroy `C_VQCMetricPlugin *metricPlugin` itself.

This function is exported in the shared library (DLL).

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

C_Init

Syntax

```
vqcStatus C_Init(vqcHDL pthis, vqcpParams *params,
                 vqcU32 *scratchBufferSize)
```

Parameters

`pthis`	Pointer to the `VQCMetricPlugin` object allocated in `VQCP_CreateMetricInstance`
`params`	Parameters for initializing the plugin
`scratchBufferSize`	Size of scratch buffer metric requests for this instance.

Description

The application calls this function to initialize a newly-created instance of the metrics plugin. The plugin should cast `pthis` to the object type corresponding with his implementation of `VQCMetricPlugin` and then call `Init()`.

This function should be declared `extern "C"`.

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

C_QueryMetricInfo

Syntax

```
vqcStatus C_QueryMetricInfo(vqcHDL pthis, vqcU32 metricId,
                            vqcpMetricInfo *metricInfo)
```

Parameters

`pthis`	Pointer to the `VQCMetricPlugin` object allocated in `VQCP_CreateMetricInstance`
`metricId`	Index of metric to query
`metricInfo`	Structure to receive info about specified metric.

Description

The application calls this function for each metric that the plugin has implemented. The total number of metrics is specified in `vqcpInfo.numMetrics`.

This function should be declared `extern "C"`.

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

C_Close

Syntax

```
vqcStatus C_Close(vqcHDL pthis)
```

Parameters

`pthis`	Pointer to the `VQCMetricPlugin` object allocated in `VQCP_CreateMetricInstance`

Description

The application calls this function to destroy a previously-created instance of the metrics plugin. The plugin should cast `pthis` to the object type corresponding with his implementation of `VQCMetricPlugin` and then call `Close()`.

This function should be declared `extern "C"`.

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

C_CalculateFrame

Syntax

```
vqcStatus C_CalculateFrame(vqcHDL pthis, vqcU32 frameIndex,
    vqcpFrameList *frameListRef, vqcpFrameList *frameListTest,
    vqcHDL scratchBuffer, vqcU32 scratchBufferSize,
    vqcF64 *frameResults, vqcF64 *avgResults);
```

Parameters

`pthis`	Pointer to the `VQCMetricPlugin` object allocated in `VQCP_CreateMetricInstance`
`frameIndex`	Index of current frame.
`frameListRef`	Array of video frames from the reference stream
`frameListTest`	Array of video frames from the test stream
`scratchBuffer`	Pointer to scratch buffer (non-persistent memory)
`scratchBufferSize`	Actual size of scratch buffer. May be less than what was requested during initialization.
`frameResults`	Array of results for each metric calculated
`avgResults`	Array of average results for each metric calculated

Description

The application calls this function to run metrics calculation on a pair of video frames (reference and test). The plugin should cast `pthis` to the object type corresponding with his implementation of `VQCMetricPlugin` and then call `CalculateFrame()`.

This function should be declared `extern “C”`.

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

VQCMetricPlugin

This class of functions consists of functions for performing metrics calculations. Each metrics plugin should create a new class which inherits `VQCMetricPlugin` and implments all of its pure virtual member functions. A new instance of this class will be created for each test stream, so it is possible to store stream-specific state information within your derived class. Multiple instances may also be created for the same test stream, each specifying a different subset of available metrics to calculate. The implementation should not attempt to share data between different instances of the same class.

Init

Syntax

```
vqcStatus Init(vqcpParams *params, vqcU32 *scratchBufferSize);
```

Parameters

`params`	Initialization parameters for the metrics plugin
`scratchBufferSize`	Size of scratch buffer metric requests for this instance.

Description

The application calls this function to create a new instance of the metrics calculation object. When multiple test streams are specified, a new instance is created for each test stream, so it is possible to store stream-specific state information.

The plugin may also request a scratch buffer, which is a single (typically large) block of memory allocated by the VQC application, which the metric may use during metrics calculation in `CalculateFrame()`. The scratch buffer is not persistent from frame to frame. Rather, it allows multiple metrics to share the same block of memory for intermediate calculations (e.g. modified copies of frames), thus reducing overall memory usage of the application.

In some situations, the application may not allocate a scratch buffer of the requested size, such as when overall memory usage is too high. Therefore each metric must check the actual scratch buffer size in each call to `CalculateFrame()` in order to know how much scratch memory is actually available.

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

Close

Syntax

```
vqcStatus Close(void);
```

Parameters

```
none
```

Description

The application calls this function to close an instance of the metrics calculation object. Any resources which were allocated during `Init()` may be freed here.

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

QueryMetricInfo

Syntax

```
vqcStatus QueryMetricInfo(vqcU32 metricId, vqcpMetricInfo *metricInfo;
```

Parameters

`metricId`	Index of metric to query
`metricInfo`	Structure to receive info about specified metric.

Description

The application calls this function for each metric that the plugin has implemented. The total number of metrics is specified in `vqcpInfo.numMetrics`. The plugin should fill in each field in the `metricInfo` structure with appropriate information about the metric.

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

CalculateFrame

Syntax

```
vqcStatus CalculateFrame(vqcU32 frameIndex,
    vqcpFrameList *frameListRef, vqcpFrameList *frameListTest,
    vqcHDL scratchBuffer, vqcU32 scratchBufferSize,
    vqcF64 *frameResults, vqcF64 *avgResults);
```

Parameters

`frameIndex`	Index of current frame.
`frameListRef`	Array of video frames from the reference stream
`frameListTest`	Array of video frames from the test stream
`scratchBuffer`	Pointer to scratch buffer (non-persistent memory)
`scratchBufferSize`	Actual size of scratch buffer. May be less than what was requested during initialization.
`frameResults`	Array of results for each metric calculated
`avgResults`	Array of average results for each metric calculated

Description

The application calls this function to calculate the metric value for a reference and test frame pair. `vqcpFrameListRef` and `vqcpFrameListTest` each contain an array of `vqcpFrame` structs, which contain pointers to the video frames for the reference and test streams, respectively. The metrics plugin fills `vqcpResultsList` with a list of `numMetrics` objects of type `vqcpResults`.

The value `frameIndex` is the the index of the current reference/test pair, i.e. the index of the frame which the results correspond to.

Note: the current version of the metrics plugin API restricts the number of frames in frameListRef and frameListTest to exactly 1. i.e. the application only provides the current frame from each stream, not future or past frames. This functionality may be added in future versions, in which case the metrics plugin would check the frameIndex member in each `vqcpFrame` object to identify the index of each frame. The maximum number of previous or future frames is determined by the corresponding fields in `vqpcMetricInfo`.

Most plugins calculate multiple metrics on each frame (e.g. separate measurement for each color component). During `Init()`, the application indicates which metrics should be calculated by sending the string `params->subscribedMetrics`. The plugin should parse this space-separated list of plugins which the application wants to have calculated, and return the results in the same order that the metrics are specified in the `subscribedMetrics` string.

`scratchBuffer` contains a pointer to a scratch buffer that the metric may use during this call to CalculateFrame(). This memory is <u>not persistent</u> from frame to frame. In some situations, the application <u>may not allocate</u> a scratch buffer of the requested size, such as when overall memory usage is too high. Therefore each metric must check the actual scratch buffer size in order to know how much scratch memory is actually available. If `scratchBufferSize` is less than was requested during initialization, the metric may only use as much memory as was actually provided.

Results for the current frame and the average of all frames are returned in frameResults and avgResults, respectively. If more than one result is returned for each metric (`vqcpMetricInfo.numResults)` these values are packed together.

<u>Note</u>: the current version only supports one results value for each metric (a frame value and average value) so `vqcpMetricInfo.numResults` must be set to 1.

Return Status

`VQC_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.0.

Structure Reference

In the following structure references, all reserved fields must be zero.

vqcpInfo

Definition

```
typedef struct {
    vqcVersion      apiVersion;
    vqcVersion      pluginVersion;
    vqcU32          position;

    const vqcChar  *pluginName;

    vqcU32          numMetrics;

    vqcU32          reserved1[16];
    vqcHDL          reserved2[4];
} vqcpInfo;
```

Description

This structure contains information about the metrics plugin. The application uses this info to register the plugin and add appropriate controls to the Metrics tab in the graphical user interface.

Members

`apiVersion`	Version of the metrics plugin API used to build the shared library. The application will skip plugins built with an incompatible version of the API (i.e. a newer metrics plugin mixed with an older version of the VQC application).
`pluginVersion`	Version of the metrics plugin. If multiple plugins with the same `className` are found in the `plugins` directory, the VQC application will only load the latest one (highest version).
`position`	Integer value to control the order in which plugins are showin the Metrics tab. Custom metrics plugins always appear at the end of the list, after the plugins provided with the application. If multiple custom metrics plugins have the same value for `position`, the application will decide the order of appearance.
`pluginName`	Name of this plugin. This is displayed in bold lettering in the Metrics tab, above the corresponding group of metrics.

`numMetrics`	The number of different metrics implemented by this plugin. During initialization, the application will inform the plugin which metrics to calculate, based on which metrics are selected/de-selected on the Metrics tab. The maximum number of metrics that can be implemented in a plugin is `MAX_NUM_METRICS = 64`.
`reserved1`	must be set to zero
`reserved2`	must be set to zero

Change History

This structure is available since SDK API 1.0.

vqcpMetricInfo

Definition

```
typedef struct {
    const vqcChar  *metrName;

    vqcF64          rangeMin;
    vqcF64          rangeMax;
    vqcF64          invalidResVal;

    vqcU32          maxFramesPrev;
    vqcU32          maxFramesFutr;
    vqcU32          numResults;
    vqcU32          complexityScale;

    vqcU32          reserved1[16];
    vqcHDL          reserved2[4];
} vqcpMetricInfo;
```

Description

This structure contains information about each metric available in the plugin.

Members

`metrName`	Name of the metric, displayed in the “Metrics” tab in the VQC application (e.g. “PSNR-Y”) The name must be in the format “metricName-chan” where chan is “Y”, “U”, “V”, or “O”. This allows each metric to be mapped correctly to the GUI controls in the Metrics tab in VQC. It is advisable to implement all 4 channels for every metricName, and return a default value (e.g. rangeMin) for any channel that is not implemented. Because hyphen (‘-‘) is used as a channel

	separator, it may not be used elsewhere in the name.
`rangeMin`	Expected minimum result value for this metric
`rangeMax`	Expected maximum result value for this metric
`invalidResVal`	Special value which plugin can specify to indicate that no result could be calculated in the call to `CalculateFrame`. Recommended default value is `NAN` from `<math.h>` but another value may also be used if desired. If set to zero, all results will be considered valid.
`maxFramesPrev`	Maximum number of previous frames the metric may access in each call to `CalculateFrame`. Must be set to zero in current version (see Known Limitations, above).
`maxFramesFutr`	Maximum number of future frames the metric may access in each call to `CalculateFrame`. Must be set to zero in current version (see Known Limitations, above).
`numResults`	Maximum number of results which this metric will return for each call to `CalculateFrame`. Must be set to one in current version (see Known Limitations, above).
`complexityScale`	Relative estimate of complexity of this metric vs. other common metrics. Must be set to zero in current version.
`reserved1`	must be set to zero
`reserved2`	must be set to zero

Change History

This structure is available since SDK API 1.0.

vqcpParams

Definition

```
typedef struct {
    vqcU32      fourCC;

    vqcU32      bpp;
    vqcU32      width[3];
    vqcU32      height[3];

    vqcU32      reserved1[16];
    vqcHDL      reserved2[4];
} vqcpParams;
```

Description

This structure contains initialization parameters which are passed to the metric plugin.

Members

`fourCC`	fourCC code indicating the format of the video frames which will be passed to the metric plugin. Note: the current version of the API passes each YUV plane separately, so fourCC will be set to zero. Chroma sampling of the video may be determined by comparing width and height of the Y plane with U/V planes.
`bpp`	Bits per pixel (same for every plane)
`width`	Frame width for each plane (0 = Y, 1 = U, 2 = V)
`height`	Frame height for each plane
`subscribedMetrics`	String containing a space-separated list of metrics that VQC wants this instance of the plugin to calculate in each call to `CalculateFrame()`. The plugin should parse this string and return results for each metric in the order specified.
`reserved1`	must be set to zero
`reserved2`	must be set to zero

Change History

This structure is available since SDK API 1.0.

vqcpFrameList

Definition

```
typedef struct {
    vqcU32      numFrames;
    vqcpFrame **frame;

    vqcU32      reserved1[16];
    vqcHDL      reserved2[4];
} vqcpFrameList;
```

Description

This structure contains an array of vcqpFrame structs, which contains the pointers to each video frame in calls to CalculateFrame.

Members

`numFrames`	Number of elements in `frameList`. In the current version of the API, this will always be 1.
`frame`	Array of frames for processing. In the current version, only frameList[0] will be valid.
`reserved1`	must be set to zero
`reserved2`	must be set to zero

Change History

This structure is available since SDK API 1.0.

vqcpFrame

Definition

```
typedef struct {
    vqcU32      frameIndex;
    vqcHDL      data[3];
    vqcU32      pitch[3];
} vqcpFrame;
```

Description

This structure describes each video frame which is passed to the plugin for metrics calculation.

Members

`frameIndex`	Index of the frame (frame 0 = first frame in the stream).
`data`	Pointers to YUV data for the frame (0 = Y, 1 = U, 2 = V). The metric plugin needs to cast each pointer to an appropriate datatype in order to access the data (e.g. vqcU8* for 8-bit content, vqcU16* for > 8-bit content).
`pitch`	Horizontal step, in bytes, for each YUV plane. This may be greater than `vqcpParams.width` due to padding.

Change History

This structure is available since SDK API 1.0.