Intel® INDE Visual Coding Framework Software Development Kit

Reference Manual

Version: Beta-1 2016

API Version 1.0

LEGAL DISCLAIMER

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Table of Contents

Overview

The Intel® INDE Visual Coding Framework (VCF) SDK (Software Development Kit) is a software development library that enables programmatic access to the VCF cross-platform runtime.

The VCF SDK is paired with the VCF Designer which allows developers to rapidly design, prototype and benchmark VCF workloads before application integration.

This document provides details on how to use the VCF API and SDK, including a complete reference for all the features of the API.

Please refer to separate VCF documentation for further details:

- VCF Release Notes – Current release limitations and known issues
- VCF User's Manual - Focusing on overall VCF architecture and VCF Designer tool

Please also refer to the VCF web page at https://software.intel.com/en-us/visual-coding-framework , for additional information about the product, downloads, FAQ and support forum.

Document Conventions

The Intel® INDE VCF SDK API 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 (`e.g. XXXXX`). All class-related items appear in all boldface, such as **VCFAPI** and **VCFGraph**. Member functions appear in boldface, such as **Init** and **Load**. Hyperlinks appear in underlined boldface, such as **`VCFAPIStatus`**.

Acronyms and Abbreviations

API	Application Programming Interface
FFmpeg*	Open source cross-platform for video/audio processing. VCF uses only muxing and demuxing capabilities of FFmpeg
SDK	Software Development Kit

Architecture

The Intel® INDE VCF SDK is a component of the greater VCF framework. The SDK provides the necessary capabilities and APIs to allow developers to integrate and deploy VCF workloads with their applications.

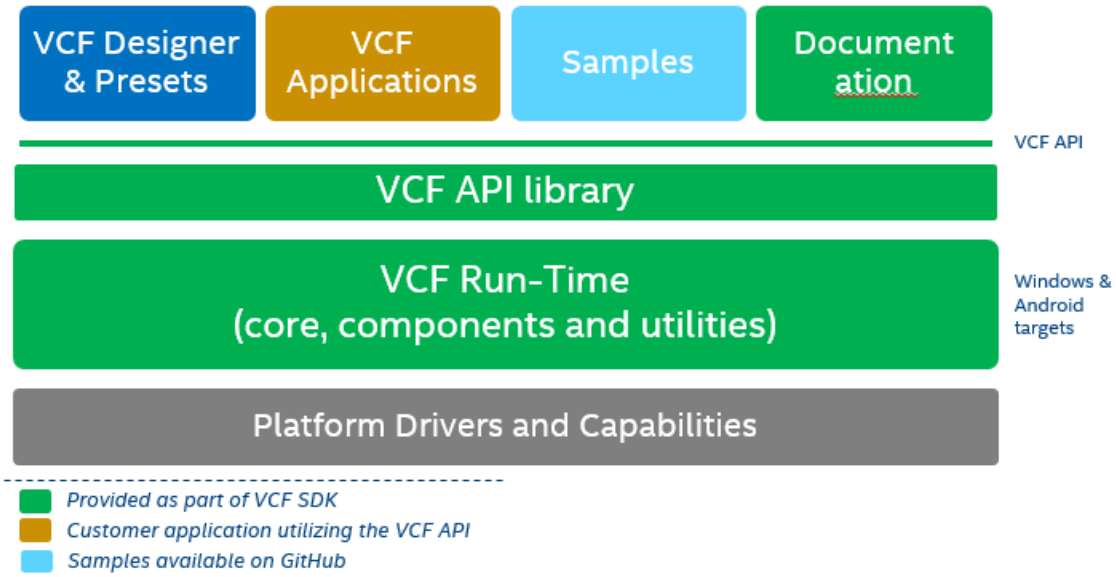


Figure 1: VCF Architecture

The components (colored "green") in the figure above are part of the VCF SDK. The VCF Designer component (colored "dark blue") is the front end user interface with which developers can develop and prototype VCF workloads. VCF sample code (colored "light blue" above) is published on GitHub* repository, https://github.com/INDExOS/visual-coding-framework.

The VCF API is grouped into the following classes:

VCFAPI	This is the top level part of the API which harbors all the run-time level functions such as run-time initialization, loading of graph workloads and run-time level event subscriptions
VCFGraph	The **VCFGraph** API provides control of individual graph workloads including operations such as Run, Stop and Pause. The API also exposes subscriptions to graph level events. Instances of **VCFGraph** API are created via the **VCFAPI**:Load or **VCFAPI**:LoadFile operation.
VCFNode	The **VCFNode** API provides control of the behavior of individual nodes. User's may subscribe for node events or invoke commands on the node.

Nodes expose a set of generic events and commands and a set of optional node specific commands and/or events.

Instances of **VCFNode** API are created via the **VCFGraph**:GetNode operation.

Callbacks All VCF APIs (**VCFAPI, VCFGraph, VCFNode)** offers a callback interface associated with subscribed events.

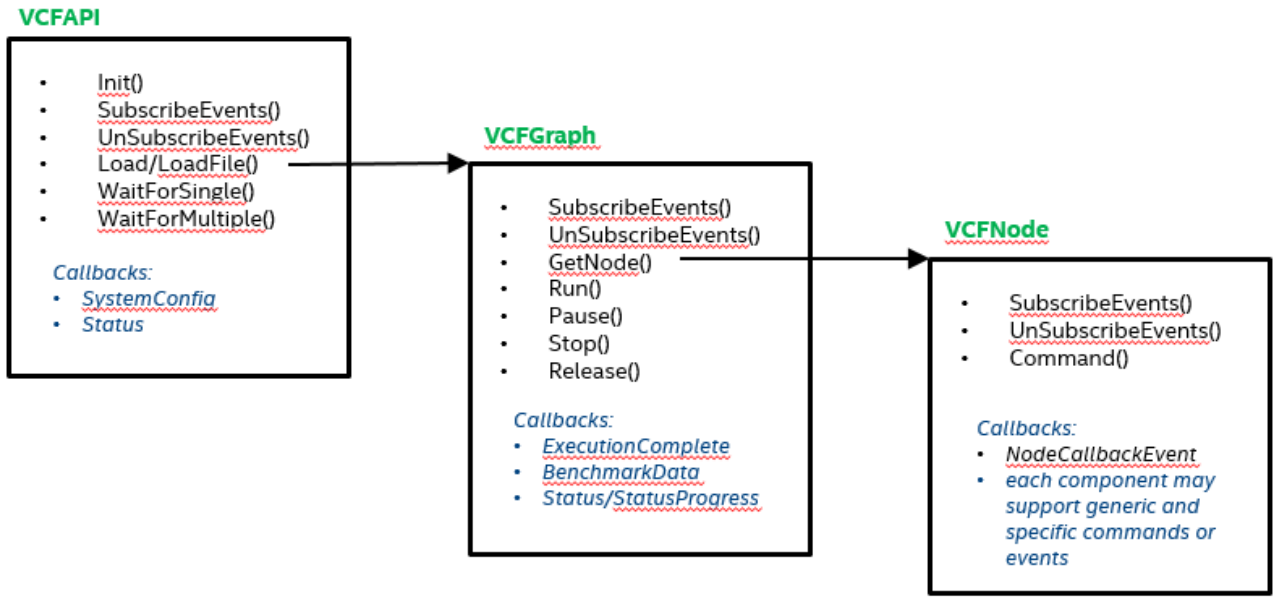


Figure 2: Overview and relationship of all the VCFAPI

The VCFAPI has language bindings for both C++ and C, as specified in the table below.

Language binding	Interface	Naming conventions

C++	vcfapi++.h	Each section of the API is represented as a separate class:
		
		`class VCFAPI`
		`class VCFGraph`
		`class VCFNode`
		
		With corresponding callbacks:
		
		`class VCFAPI_Callback`
		`class VCFGraph_Callback`
		`class VCFNode_Callback`

C	vcfapi.h	Operations are prefixed with the corresponding API section name:
		
		`VCFAPI_<operation name>`
		`VCFGraph_<operation name>`
		`VCFNode_<operation name>`
		
		With corresponding callback signatures:
		
		`void (*t_VCFAPICallback)(VCFAPIEventMask event, void* data);`
		`void (*t_VCFGraphCallback)(vcfGraphHandle graphHandle, VCFGraphEventMask event, void* data);`
		`void (*t_VCFNodeCallback)(const VCFNodeEventContext* evtCtx, void* data);`

Applications utilizing the VCF API must link with the static VCF run-time API library.

The VCF run-time consists of multiple re-distributable objects. Depending application workload, all or a subset of the objects are distributed with the customer application. The below table lists all the VCF re-distributable run-time objects.

Run-time capability	Windows Re-distributables	Android Re-distributables
FFmpeg* features	avcodec<version>.dll avformat<version>.dll avutil<version>.dll	libavcodec-<version>.so libavformat-<version>.so libavutil-<version>.so
Core VCF features	vcfrtcore.dll fgd_utils.dll graph.dll plugin_manager.dll xml_parser.dll fgd_graphml_reader.dll fgd_rule_check.dll	libvcfrtcore.so libfgd_utils.so libgraph.so libplugin_manager.so libxml_parser.so libfgd_graphml_reader.so libfgdrules.so libandroid_environment.so
Software codecs (VideoDecode/ Encode/Process and AudioDecode/ Encode nodes)	libmfxsw<arch>.dll libmfxaudiosw<arch>.dll	NA

Intel® Threading Building blocks	tbb.dll	libtbb.so
VCF components (only need to distribute the set of nodes used by desired workload)	vcffilesource.dll vcfvideodecode.dll vcfrender.dll etc.	libnode_vcffilesource.so libnode_vcfvideodecode.so libnode_vcfrender.so etc.
VCF utilities (only need to distribute the set of utilities used by desired workload)	vcfbuffermemmanager.dll vcfdevicemanager.dll etc.	libutilities_vcfbuffermemorymanager.so libutilities_vcfdevicemanager.so etc.

A utility is available to determine the minimum set of required objects (DLL/so) required for a specific workload. Use the utility as follows:

```
depcheck.exe [32|64] [graphml file name] [windows|android] [module copy folder]
```

Example: Determine required set objects for my.graphml workload for Windows (32 bit OS).

```
depcheck.exe 32 myfolder\my.graphml windows
```

Programming Guide

This chapter describes the concepts and available operations of the VCF SDK.

Since VCF supports multiple language bindings, the function signature of the individual API operation varies depending on the language binding used by the application. The examples provided in this document focus on the C++ language bindings. Please refer to the SDK include files for signatures for other language bindings.

Below is a simple C++ code sample, which utilizes the VCF API to load and run a VCF graph workload (for simplicity, the sample ignores all VCF status codes error handling).

```
VCFAPI myVCFAPI; // Create VCF run-time instance

myVCFAPI.Init(); // Initialize VCF runtime

VCFGraph* myGraph;
myVCFAPI.LoadFile("workload.graphml", &myGraph); // Load and validate VCF graph

myGraph->Run(); // Execute the graph

myVCFAPI.WaitForSingle(myGraph); // Wait for graph execution to complete

myGraph->Release(); // Release all resources associated with the graph workload
```

The following chapters describe the API operations used in the above sample and the rest of the VCF API.

Status Codes

The Intel® VCF SDK APIs has a unified set of status codes (`VCFAPIStatus`), which conveys the result of API operations. See below for the list of status codes with generic description.

Status Code	Description
`VCFAPI_STATUS_SUCCESS`	API operation was successful
`VCFAPI_STATUS_ERR_UNKNOWN`	Unknown or unspecified error
`VCFAPI_STATUS_ERR_UNSUPPORTED`	Unsupported feature
`VCFAPI_STATUS_ERR_MEMORY_ALLOC`	Out of memory
`VCFAPI_STATUS_ERR_ABORTED`	Operation aborted

VCFAPI_STATUS_ERR_RT_NOT_FOUND	VCF run-time was not located
VCFAPI_STATUS_ERR_INVALID_GRAPH	Invalid VCF graph
VCFAPI_STATUS_ERR_NOT_INITIALIZED	Feature not initialized
VCFAPI_STATUS_ERR_NOT_FOUND	Object or reference not found
VCFAPI_STATUS_ERR_INVALID_OPERATION	Invalid operation
VCFAPI_STATUS_ERR_TIMEOUT	Operation timeout
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid NULL pointer input
VCFAPI_STATUS_ERR_INVALID_VALUE	Invalid input value
VCFAPI_STATUS_WRN_UNSPECIFIED	Unspecified warning. TBD

For more verbose error details, please subscribe to `VCFAPI_EVENTMASK_STATUS_CORE_ERROR` and/or `VCFGRAPH_EVENTMASK_STATUS_ERROR.`

Each API operation has a set possible status codes. Refer to following chapters for details.

VCF status code include file: “vcfapi_def.h”.

VCFAPI Lifetime

Before calling any SDK functions, the application must initialize the VCF run-time by calling VCFAPI:Init() (or VCFAPI_Init() for C). This operation finds the VCF run-time and prepares it for use by the rest of the VCF API.

If needed, users can create multiple instances of VCFAPI on the application stack or heap. All instances of VCFAPI in the same application process shares the same backend VCF run-time instance.

Observe that applications must ensure that the VCFAPI instance is destructed/closed on the same application thread as it was created.

Runtime level operations

The run-time level part of the VCFAPI, denoted **VCFAPI**, is the top level API, which harbors all the run-time level functions such as run-time initialization, loading of graph workloads and run-time level event subscriptions. Below table lists the available **VCFAPI** operations.

C++	C
Init	`VCFAPI_Init`

SubscribeEvents	VCFAPI_SubscribeEvents
UnSubscribeEvents	VCFAPI_UnSubscribeEvents
LoadFile	VCFAPI_LoadFile
Load	VCFAPI_Load
WaitForSingle	VCFAPI_WaitForSingle
WaitForMultiple	VCFAPI_WaitForMultiple
	VCFAPI_Close

Please refer to the following chapters for detailed usage for each operation.

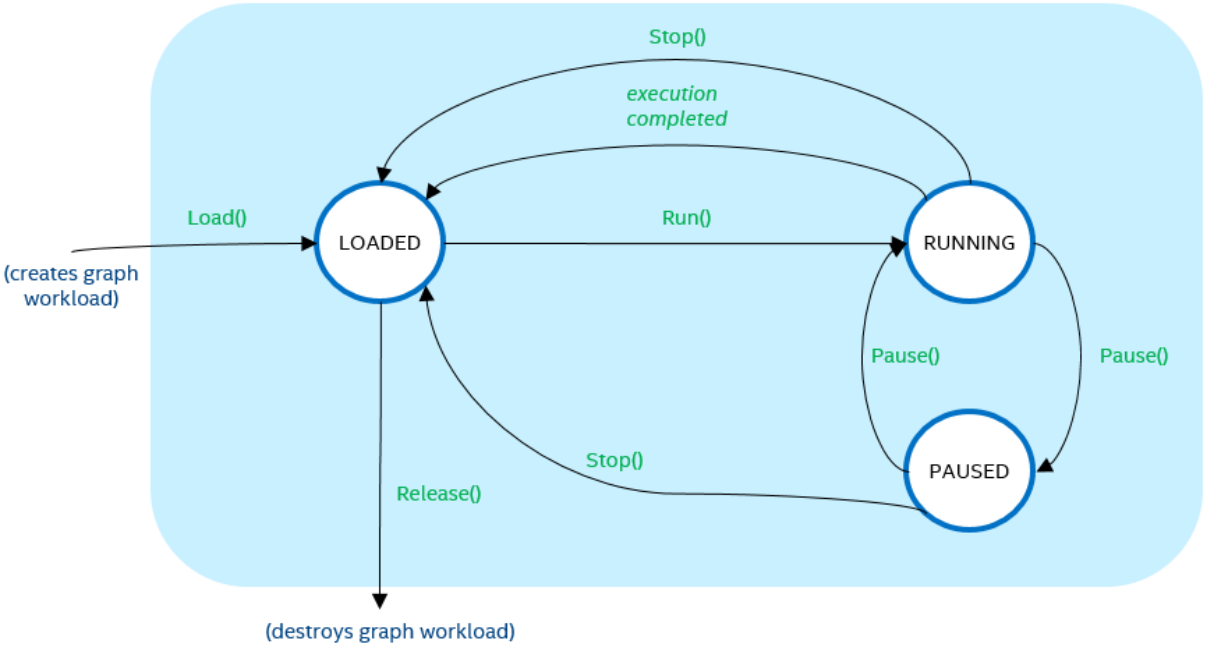
Graph level operations

The **VCFGraph** API controls individual graph workloads including operations such as Run, Stop and Pause. The API also exposes subscriptions to graph level events. Below table lists the available **VCFGraph** operations.

C++	C
SubscribeEvents	VCFGraph_SubscribeEvents
UnSubscribeEvents	VCFGraph_UnSubscribeEvents
Run	VCFGraph_Run
Stop	VCFGraph_Stop
Pause	VCFGraph_Pause
Release	VCFGraph_Release
GetNode	VCFGraph_GetNode

Please refer to the following chapters for detailed usage for each operation.

Each graph workload has a state machine, which governs the behavior of the workload and available operations for each workload state. The following figure shows the workload state machine.



Node level operations

The **VCFNode** API controls the behavior of individual nodes. User's may subscribe for node events or invoke commands on the node. Below table lists the available **VCFNode** operations.

C++	C
SubscribeEvents	VCFNode_SubscribeEvents
UnSubscribeEvents	VCFNode_UnSubscribeEvents
Command	VCFNode_Command

Please refer to the following chapters for detailed usage for each operation.

Graph and Node naming conventions

As a graph is created in the VCF Designer tool, each node is assigned a unique name. To retrieve a handle to the node use the `VCFGraph:GetNode` operation.

Nodes have the following naming convention:

```
<node type name>_<platform identifier>_node_<ID>
```

Where <node type name> denotes the node type such as "Video Decode", which is a built-in VCF node with the name "VCFVideoDecode".

<platform identifier> denotes the selected node platform capability type. Three values are currently supported:

- "C" : Compatibility node. Node supports both "W" and "A"
- "W" : Windows node. Node supports Windows specific features
- "A" : Android node. Node supports Android specific features

<ID> denotes the unique node identifier.

Example unique node name:

```
VCFVideoDecode_W_node_5
```

To retrieve a handle for access to graph level properties, use the `VCFGraph::GetNode` call with `nodeName` value set to `"General"`.

Function Reference

This section describes SDK functions and their operations.

Each function is described as follows:

- The header of the below chapters show the function name as follows: <C++ class member name> / <C function name>
- The "Syntax" section shows the function signature of the C++ member function. For simplicity, the C++ signature is used throughout.
- The "Parameters" section lists all the input and/or output parameters exposed by the function
- The "Description" section provides a detailed description of the function and its purpose. Potential side effects listed, if any.
- The "Return Status" lists the set of possible return codes and how to interpret them.

VCFAPI Functions

The following chapters details all of the available **VCFAPI** functions.

Init / VCFAPI_Init

Syntax

```
VCFAPIStatus Init(VCFAPIConfig* pConfig = 0);
```

Parameters

pConfig	Run-time configuration options. On Windows, this parameter is currently reserved! Do not use. On Android, this parameter is used to send Android Environment related items to the run-time. See: VCFAPIConfigItemAndroidEnv.

Description

This function initializes the VCF run-time.

Observe that application must ensure that the VCFAPI instance is destructed/closed on the same application thread as it was created.

Return Status

VCFAPI_STATUS_SUCCESS	The opersation completed successfully

VCFAPI_STATUS_ERR_RT_NOT_FOUND	VCF run-time was not found. Make sure that "vcfrtcore.dll" or "libvcfrtcore.so" is located in the same folder as the application executable.

VCFAPI_Close

Syntax

```
void VCFAPI_Close();
```

Parameters

none

Description

This function releases the VCF run-time resources.

Only applicable to C language use cases. When using C++, this operation is invoked automatically when the **VCFAPI** object is destructed.

Observe that application must ensure that the VCFAPI instance is destructed/closed on the same application thread as it was created.

Return Status

none

SubscribeEvents / VCFAPI_SubscribeEvents

Syntax

```
VCFAPIStatus SubscribeEvents(unsigned int eventMask,
                             VCFAPI_Callback* pCB);
```

Parameters

eventMask	Event mask for the selected set of events
pCB	Callback object which implements the VCFAPI_Callback interface

Description

Use this function to subscribe to VCF run-time level events. The set of available events are defined in VCFAPIEventMask (located in "vcfapi.h")

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time is not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid callback input pointer
VCFAPI_STATUS_ERR_UNSUPPORTED	Unsupported event type
VCFAPI_STATUS_ERR_NOT_FOUND	Failed to find or initialize internal VCF run-time

UnSubscribeEvents / VCFAPI_UnSubscribeEvents

Syntax

```
VCFAPIStatus UnSubscribeEvents(unsigned int eventMask,
                               VCFAPI_Callback* pCB);
```

Parameters

eventMask	Event mask for the selected set of events
pCB	Callback object which implements the VCFAPI_Callback interface

Description

Use this function to un-subscribe VCF run-time level events. The set of available events are defined in VCFAPIEventMask (located in "vcfapi.h")

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid callback input pointer
VCFAPI_STATUS_ERR_UNSUPPORTED	Unsupported event type
VCFAPI_STATUS_ERR_NOT_FOUND	Failed to find or initialize internal VCF run-time

LoadFile / VCFAPI_LoadFile

Syntax

```
VCFAPIStatus LoadFile(const char* graphFile,
                      VCFGraph** ppGraph);
```

Parameters

`graphFile`	File name of the graph to be loaded
`ppGraph`	Returns the **VCFGraph** interface handle

Description

This function loads a VCF workload (specified as GraphML) from a file and returns a **VCFGraph** interface handle.

The process of loading a graph involves multiple steps, such as graph validation (e.g. invalid syntax, node types or connections) and instantiation of a complete graph model required for execution.

Return Status

`VCFAPI_STATUS_SUCCESS`	The operation completed successfully. The `ppGraph` (**VCFGraph)** handle is valid
`VCFAPI_STATUS_ERR_NOT_INITIALIZED`	VCF run-time not initialized
`VCFAPI_STATUS_ERR_NULL_POINTER`	Invalid input pointer
`VCFAPI_STATUS_ERR_MEMORY_ALLOC`	Out of memory while loading graph
`VCFAPI_STATUS_ERR_INVALID_GRAPH`	Graph is invalid or does not comply with VCF graph requirements
`VCFAPI_STATUS_ERR_NOT_FOUND`	Failed to find or initialize VCF run-time or graph node

Load / VCFAPI_Load

Syntax

```
VCFAPIStatus Load(const char* graph,
                  VCFGraph** ppGraph);
```

Parameters

`graph`	Full string representation of the graph to be loaded

`ppGraph`	Returns the **VCFGraph** interface handle

Description

This function loads a VCF workload (specified as GraphML) from a file and returns a **VCFGraph** interface handle.

The process of loading a graph involves multiple steps, such as graph validation (e.g. invalid syntax, node types or connections) and instantiation of a complete graph model required for execution.

Return Status

`VCFAPI_STATUS_SUCCESS`	The operation completed successfully. The `ppGraph` (**VCFGraph)** handle is valid
`VCFAPI_STATUS_ERR_NOT_INITIALIZED`	VCF run-time not initialized
`VCFAPI_STATUS_ERR_NULL_POINTER`	Invalid input pointer
`VCFAPI_STATUS_ERR_MEMORY_ALLOC`	Out of memory while loading graph
`VCFAPI_STATUS_ERR_INVALID_GRAPH`	Graph is invalid or does not comply with VCF graph requirements
`VCFAPI_STATUS_ERR_NOT_FOUND`	Failed to find or initialize VCF run-time or graph node

WaitForSingle / VCFAPI_WaitForSingle

Syntax

```
VCFAPIStatus WaitForSingle(VCFGraph* pGraph,
                           unsigned int timeoutSec = 0);
```

Parameters

`pGraph`	**VCFGraph** handle representing the workload
`timeoutSec`	Timeout in seconds. If not set, timeout is not defined (=infinite)

Description

This function waits for a single VCF graph (identified by a **VCFGraph** handle) to complete execution.

The "timeoutSec" parameter allows specifying an optional time limit, within which the graph must execute. If the graph does not complete execution within the specified time, the workload is aborted and the function returns VCFAPI_STATUS_ERR_TIMEOUT.

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully. Workload execution completed
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer
VCFAPI_STATUS_ERR_INVALID_OPERATION	Not a valid operation. For instance, this will occur if the workload has not yet been started (**VCFGraph:Run()**).
VCFAPI_STATUS_ERR_TIMEOUT	Graph did not complete execution within the specified time limit.
VCFAPI_STATUS_ERR_NOT_FOUND	**VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has been released (**VCFGraph:Release()**)

WaitForMultiple / VCFAPI_WaitForMultiple

Syntax

```
VCFAPIStatus WaitForMultiple(VCFGraph** ppGraph,
                             unsigned int numGraphs,
                             unsigned int timeoutSec = 0);
```

Parameters

ppGraph	Pointer to array of **VCFGraph** handles representing the workloads the user wants to wait for
numGraphs	The number of graphs in the array
timeoutSec	Timeout in seconds. If not set, timeout is not defined (=infinite)

Description

This function waits for multiple VCF graphs (identified with a **VCFGraph** handle) to complete execution.

The "timeoutSec" parameter allows specifying an optional time limit, within which the graph must execute. If the graph does not complete execution within the specified time, the workload is aborted and the function returns VCFAPI_STATUS_ERR_TIMEOUT.

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully. Workload execution completed
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer
VCFAPI_STATUS_ERR_INVALID_OPERATION	Not a valid operation. For instance, this will occur if the workload has not yet been started (**VCFGraph:Run()**).
VCFAPI_STATUS_ERR_TIMEOUT	Graph did not complete execution within the specified time limit.
VCFAPI_STATUS_ERR_NOT_FOUND	**VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has been released (**VCFGraph:Release()**)

VCFGraph Functions

The following chapters details all of the available **VCFGraph** functions.

SubscribeEvents / VCFGraph_SubscribeEvents

Syntax

```
VCFAPIStatus SubscribeEvents(unsigned int eventMask,
                             VCFGraph_Callback* pCB);
```

Parameters

eventMask	Event mask for the selected set of events
pCB	Callback object which implements the VCFGraph_Callback interface

Description

Use this function to subscribe to VCF graph level events. The set of available events are defined in VCFGraphEventMask (located in "vcfapi.h")

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer
VCFAPI_STATUS_ERR_UNSUPPORTED	Unsupported event type
VCFAPI_STATUS_ERR_NOT_FOUND	**VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has been released (**VCFGraph:Release()**)

UnSubscribeEvents / VCFGraph_UnSubscribeEvents

Syntax

```
VCFAPIStatus UnSubscribeEvents(unsigned int eventMask,
                               VCFGraph_Callback* pCB);
```

Parameters

eventMask	Event mask for the selected set of events
pCB	Callback object which implements the VCFGraph_Callback interface

Description

Use this function to un-subscribe to VCF graph level events. The set of available events are defined in VCFGraphEventMask (located in "vcfapi.h")

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer
VCFAPI_STATUS_ERR_UNSUPPORTED	Unsupported event type
VCFAPI_STATUS_ERR_NOT_FOUND	**VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has been released (**VCFGraph:Release()**)

Run / VCFGraph_Run

Syntax

```
VCFAPIStatus Run();
```

Parameters

None

Description

This function starts execution of a VCF graph. The function is non-blocking and returns immediately. To block application execution until the VCF graph completes, use the **VCFAPI:WaitForSingle** or **VCFAPI:WaitForMultiple** functions.

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer (only applies to C language binding)
VCFAPI_STATUS_ERR_INVALID_OPERATION	Not a valid operation. For instance, this will occur if trying to execute a graph that is already in executing state.
VCFAPI_STATUS_ERR_NOT_FOUND	Associated **VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has been released (**VCFGraph:Release()**)

Pause / VCFGraph_Pause

Syntax

```
VCFAPIStatus Pause(VCFGraphState* pState = 0);
```

Parameters

pState	Optionally returns the new workload state after invoked Pause operation.

	The parameter has two valid values: VCFGRAPH_STATE_RUNNING or VCFGRAPH_STATE_PAUSED

Description

This function pauses or un-pauses a currently executing graph. The function is non-blocking and returns immediately.

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer (only applies to C language binding)
VCFAPI_STATUS_ERR_INVALID_OPERATION	Not a valid operation. For instance, this will occur if trying to pause a graph which is not executing
VCFAPI_STATUS_ERR_NOT_FOUND	Associated **VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has been released (**VCFGraph:Release()**)

Stop / VCFGraph_Stop

Syntax

```
VCFAPIStatus Stop();
```

Parameters

```
none
```

Description

This function stops graph execution. The function is non-blocking and returns immediately.

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully

VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer (only applies to C language binding)
VCFAPI_STATUS_ERR_INVALID_OPERATION	Not a valid operation. For instance, this will occur if trying to stop a graph which is not executing
VCFAPI_STATUS_ERR_NOT_FOUND	Associated **VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has been released (**VCFGraph:Release()**)

Release / VCFGraph_Release

Syntax

```
VCFAPIStatus Release();
```

Parameters

```
none
```

Description

This function releases all the resources associated with a graph.

If the graph has not yet completed execution, the graph will be automatically stopped, then released.

Note that the graph must be released in the same thread as it was created (corresponding to the **VCFAPI:Load/LoadFile()** call).

All calls using the **VCFGraph** handle after release are invalid.

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer (only applies to C language binding)
VCFAPI_STATUS_ERR_INVALID_OPERATION	Not a valid operation. For instance, release invoked in other thread compared to the thread in which the graph was loaded (**VCFAPI:Load/LoadFile()**)

VCFAPI_STATUS_ERR_NOT_FOUND	Associated **VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has already been released (**VCFGraph:Release()**)

GetNode / VCFGraph_GetNode

Syntax

```
VCFAPIStatus GetNode(const char* nodeName,
                     VCFNode** ppNode);
```

Parameters

nodeName	Name of a valid node part of the VCF graph.
ppNode	Returns the **VCFNode** interface handle

Description

This function retrieves a node handle (**VCFNode** interface) to a node part of the graph. Please refer to the VCF node level functions for details about the **VCFNode** interface.

To retrieve a handle for access to graph level properties, use the `GetNode` call with `nodeName` value set to `"General"`.

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer
VCFAPI_STATUS_ERR_NOT_FOUND	Associated **VCFGraph** handle not found. This will occur if invalid handle is used or if the graph has already been released (**VCFGraph:Release()**) or if trying to gain access to a node which is NOT part of the graph.

VCFNode Functions

The following chapters details all of the available **VCFNode** functions.

SubscribeEvents / VCFNode_SubscribeEvents

Syntax

```
VCFAPIStatus SubscribeEvents(unsigned int eventMask,
                             VCFNode_Callback* pCB);
```

Parameters

eventMask	Event mask for the selected set of events
pCB	Callback object which implements the VCFNode_Callback interface

Description

Use this function to subscribe to VCF node level events. The set of available events are defined in VCFGenericComponentEvents (located in "vcfapi_control.h"). Node specific events such as VCFDecodeComponentEvents (located in "vcfapi_control.h").

Return Status

VCFAPI_STATUS_SUCCESS	The operation completed successfully
VCFAPI_STATUS_ERR_NOT_INITIALIZED	VCF run-time not initialized
VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer
VCFAPI_STATUS_ERR_NOT_FOUND	Associated **VCFGraph** or **VCFNode** handle not found. This will occur if invalid handle is used or if the graph has already been released (**VCFGraph:Release()**)

UnSubscribeEvents / VCFNode_UnSubscribeEvents

Syntax

```
VCFAPIStatus UnSubscribeEvents(unsigned int eventMask,
                               VCFNode_Callback* pCB);
```

Parameters

eventMask	Event mask for the selected set of events
pCB	Callback object which implements the VCFNode_Callback interface

Description

Use this function to un-subscribe to VCF node level events. The set of available events are defined in `VCFGenericComponentEvents` (located in "vcfapi_control.h"). Node specific events such as `VCFDecodeComponentEvents` (located in "vcfapi_control.h").

Return Status

`VCFAPI_STATUS_SUCCESS`	The operation completed successfully
`VCFAPI_STATUS_ERR_NOT_INITIALIZED`	VCF run-time not initialized
`VCFAPI_STATUS_ERR_NULL_POINTER`	Invalid input pointer
`VCFAPI_STATUS_ERR_NOT_FOUND`	Associated **VCFGraph** or **VCFNode** handle not found. This will occur if invalid handle is used or if the graph has already been released (**VCFGraph:Release()**)

Command / VCFNode_Command

Syntax

```
VCFAPIStatus Command(unsigned int commandId,
                     void* pData = 0);
```

Parameters

`commandId`	Identifier for the node command that should be executed
`pData`	Pointer to data object representing the selected command Pointer may be NULL for commands that do not need to convey any data to the VCF node

Description

This function invokes a command on the node. The set of available commands vary depending on the specific node.

Nodes may support any of the commands specified in `VCFGenericComponentCommands`. Nodes may also implement commands that are node specific, such as **`VCFNODE_COMMAND_RENDER_SET_TARGET`**. Note that all node specific commands must have an ID with offset **`VCFNODE_COMMAND_CUSTOM_BASE`**

Return Status

`VCFAPI_STATUS_SUCCESS`	The operation completed successfully
`VCFAPI_STATUS_ERR_NOT_INITIALIZED`	VCF run-time not initialized

VCFAPI_STATUS_ERR_NULL_POINTER	Invalid input pointer (only applies to C language binding)
VCFAPI_STATUS_ERR_NOT_FOUND	Associated **VCFGraph** or **VCFNode** handle not found. This will occur if invalid handle is used or if the graph has already been released (**VCFGraph:Release()**) This status code may apply to other scenarios where a particular resource is not found.

VCFAPI Callback Interface

Subscribing to API events (realized as callbacks) is optional. To subscribe for **VCFAPI** (run-time level) events, user implements a callback interface and provide the pointer to the implementation to VCF via the **VCFAPI:SubscribeEvents()** call.

For the C++ API, the event callbacks is exposed via the `VCFAPI_Callback` interface.

The event callback interface for the C API is exposed a as a generic function signature:

```
typedef void (*t_VCFAPICallback)(VCFAPIEventMask event, void* data);
```

The following chapters detail all of the available run-time level callback events.

SystemConfig

Syntax

```
void SystemConfig(const char* component,
                  const char* name,
                  const char* value,
                  const char* attribute);
```

Parameters

component	Origination. Can be "General" if applicable to workload as a whole, or node name if data applies to specific node only.
Name	Name of the characteristic
Value	The value of the characteristic
Attribute	Additional attributes (optional)

Description

This callback returns information about a platform and environment capabilities, such as Processor type or Graphics driver version.

To subscribe to this callback use the **VCFAPIEventMask:VCFAPI_EVENTMASK_SYSTEM_CONFIG** event ID.

To retain the data returned by **SystemConfig** user must copy the data before returning from the function.

Return Status

None

Status

Syntax

```
void Status(const char* status,
            const int res);
```

Parameters

status	Status message
res	Error code associated with the event (optional)

Description

This callback returns verbose status (errors and info) about VCF run-time.

To subscribe to this callback use any combination of the followng event IDs:
VCFAPIEventMask:VCFAPI_EVENTMASK_STATUS_CORE_ERROR
VCFAPIEventMask:VCFAPI_EVENTMASK_STATUS_CORE_INFO

To retain the data returned by **Status** user must copy the data before returning from the function.

Return Status

none

VCFGraph Callback Interface

Subscribing to API events (realized as callbacks) is optional. To subscribe for **VCFGraph** (graph level) events, user implements a callback interface and provide the pointer to the implementation to VCF via the **VCFGraph:SubscribeEvents()** call.

For the C++ API, the event callbacks is exposed via the `VCFGraph_Callback` interface.

The event callback interface for the C API is exposed a as a generic function signature:
```
typedef void (*t_VCFGraphCallback)(vcfGraphHandle graphHandle, VCFGraphEventMask event, void* data);
```

The following chapters detail all of the available graph level callback events.

Status

Syntax

```
void Status(VCFGraph* pGraph,
            const char* component,
            const char* status,
            const int res);
```

Parameters

pGraph	Originating VCF graph handle
component	Name of originating VCF node/component
status	Status message
res	Error code associated with the event (optional)

Description

This callback returns verbose status (errors, warnings and info) about that graph workload.

To subscribe to this callback use any combination of the followng event IDs:
VCFGraphEventMask:VCFGRAPH_EVENTMASK_STATUS_ERROR
VCFGraphEventMask:VCFGRAPH_EVENTMASK_STATUS_WARNING
VCFGraphEventMask:VCFGRAPH_EVENTMASK_STATUS_INFO

To retain the data returned by **Status** user must copy the data before returning from the function.

Return Status

none

StatusProgress

Syntax

```
void StatusProgress(VCFGraph* pGraph,
                    const char* sourceComponent,
                    unsigned int numPackets);
```

Parameters

pGraph	Originating VCF graph handle
sourceComponent	Name of originating VCF source node/component
numPackets	Total number of packets processed

Description

This callback reports graph execution progress. Progress is reported at ~1 second interval.

To subscribe to this callback use the followng event ID:
VCFGraphEventMask:VCFGRAPH_EVENTMASK_STATUS_PROGRESS

If graph has multiple source nodes, then each of the source nodes will report progress individually (**sourceComponent** conveys origin of the callback).

To retain the data returned by **StatusProgress** user must copy the data before returning from the function.

Return Status

None

ExecutionComplete

Syntax

```
void ExecutionComplete(VCFGraph* pGraph,
                       VCFGraphExecutionCompleteStatus sts);
```

Parameters

pGraph	Originating VCF graph handle
sts	Status code indicating the reason for completion. Possible values of are defined in VCFGraphExecutionCompleteStatus

Description

This callback is invoked when the graph has been completely executed, aborted or stopped.

To subscribe to this callback use the followng event ID:
VCFGraphEventMask:VCFGRAPH_EVENTMASK_EXECUTION_COMPLETE

Return Status

None

BenchmarkData

Syntax

```
void BenchmarkData(VCFGraph* pGraph,
                   const char* data);
```

Parameters

pGraph	Originating VCF graph handle
data	String containing interleaved key/value benchmark data pairs

Description

This callback delivers high level graph execution benchmarks and is invoked after graph workload has completed.

To subscribe to this callback use the followng event ID:
VCFGraphEventMask:VCFGRAPH_EVENTMASK_BENCHMARK_DATA

The received data will depend on GraphML workload configuration.

To retain the data returned by **BenchmarkData** user must copy the data before returning from the function.

Return Status

none

VCFNode Callback Interface

Subscribing to API events (realized as callbacks) is optional. To subscribe for **VCFNode** (node level) events, user implements a callback interface and provide the pointer to the implementation to VCF via the **VCFNode:SubscribeEvents()** call.

For the C++ API, the event callbacks are exposed as a generic function signature, `VCFNode_Callback.` A generic signature allows flexible extension for custom node events.

The callback interface for the C API is also exposed a as a generic function signature:

```
typedef void (*t_VCFNodeCallback)(const VCFNodeEventContext* evtCtx, void* data);
```

The following chapters detail the currently available set of node level callback events.

NodeCallbackEvent

Syntax

```
void NodeCallbackEvent(const VCFNodeEventContext* evtCtx,
                       void* pData);
```

Parameters

evtCtx	Event context. Contains details about the event origination and event ID. Please refer to VCFNodeEventContext for details.
pData	Pointer to event specific data. Pointer must be casted to corresponding event data type.

Description

This callback is invoked for all node level events. User must inspect the **eventId** member of the received VCFNodeEventContext (**evtCtx**) to determine the originating event.

Based on the event ID, the user must cast the received **pData** (void*) pointer to the corresponding event data structure.

To subscribe to this callback use any combination of events from VCFGenericComponentEvents and/or node/component specific events such as VCFDecodeComponentEvents.

To retain the data returned by **NodeCallbackEvent** user must copy the data before returning from the function.

Return Status

none

Data Structure Reference

This chapter describe the SDK data structures referenced in the previous chapters.

VCFAPIEventMask

```
typedef enum {
    VCFAPI_EVENTMASK_SYSTEM_CONFIG          = 0x1,
    VCFAPI_EVENTMASK_STATUS_CORE_INFO       = 0x2,
    VCFAPI_EVENTMASK_STATUS_CORE_ERROR      = 0x4,
    VCFAPI_EVENTMASK_ALL                    = 0x7
} VCFAPIEventMask;
```

The VCFAPI_Callback (C++ API) interface is invoked when one of the above events occur.

Identifier name	Purpose
VCFAPI_EVENTMASK_SYSTEM_CONFIG	Subscribe for system configuration, including information such as processor details. C language binding data structure: VCFAPICallbackDataSystemConfig
VCFAPI_EVENTMASK_STATUS_CORE_INFO	Subscribe for execution status info. C language binding data structure: VCFAPICallbackDataStatus
VCFAPI_EVENTMASK_STATUS_CORE_ERROR	Subscribe for execution status errors. C language binding data structure: VCFAPICallbackDataStatus

VCFGraphEventMask

```
typedef enum {
    VCFGRAPH_EVENTMASK_STATUS_ERROR         = 0x1,
    VCFGRAPH_EVENTMASK_STATUS_WARNING       = 0x2,
    VCFGRAPH_EVENTMASK_STATUS_INFO          = 0x4,
    VCFGRAPH_EVENTMASK_STATUS_PROGRESS      = 0x8,
    VCFGRAPH_EVENTMASK_EXECUTION_COMPLETE   = 0x10,
    VCFGRAPH_EVENTMASK_BENCHMARK_DATA       = 0x20,
    VCFGRAPH_EVENTMASK_ALL                  = 0x3F
} VCFGraphEventMask;
```

The VCFGraph_Callback (C++ API) interface is invoked when one of the above events occur.

Identifier name	Purpose
VCFGRAPH_EVENTMASK_STATUS_ERROR	Subscribe for execution status errors. C language binding data structure: VCFGraphCallbackDataStatus
VCFGRAPH_EVENTMASK_STATUS_WARNING	Subscribe for execution status warnings. C language binding data structure: VCFGraphCallbackDataStatus

VCFGRAPH_EVENTMASK_STATUS_INFO	Subscribe for execution status info. C language binding data structure: VCFGraphCallbackDataStatus
VCFGRAPH_EVENTMASK_STATUS_PROGRESS	Subscribe for execution progress C language binding data structure: VCFGraphCallbackDataStatusProgress
VCFGRAPH_EVENTMASK_EXECUTION_COMPLETE	Subscribe for graph execution complete event. C language binding data structure: VCFGraphCallbackDataExecutionComplete
VCFGRAPH_EVENTMASK_BENCHMARK_DATA	Subscribe for benchmark data. Data is collected after workload has completed execution. C language binding data structure: VCFGraphCallbackDataBenchmark

VCFGenericComponentEvents

```
typedef enum {
    VCFNODE_EVENT_EXECUTION_COMPLETED   = 0x01,
    VCFNODE_EVENT_ACTUAL_CONFIG         = 0x02,
    VCFNODE_EVENT_CUSTOM_BASE           = 0x10000
} VCFGenericComponentEvents;
```

The VCFNode_Callback (C++ API) interface is invoked when one of the above events occur.

All custom node events must be offset by VCFNODE_EVENT_CUSTOM_BASE

Identifier name	Purpose
VCFNODE_EVENT_EXECUTION_COMPLETED	Subscribe for node execution completion event. Associated data structure: VCFNodeCallbackDataExecutionComplete
VCFNODE_EVENT_ACTUAL_CONFIG	Subscribe for actual configuration of node. Associated data structure: VCFNodeCallbackDataActualConfig
VCFNODE_EVENT_CUSTOM_BASE	<Base for custom node events>

VCFGenericComponentCommands

```
typedef enum {
    VCFNODE_COMMAND_CHANGE_PARAM_VALUE  = 1,
    VCFNODE_COMMAND_GET_PARAM_VALUE     = 2,
    VCFNODE_COMMAND_DISABLE             = 3,
    VCFNODE_COMMAND_ENABLE              = 4,
    VCFNODE_COMMAND_CUSTOM_BASE         = 1000
} VCFGenericComponentCommands;
```

All custom node commands must be offset by VCFNODE_COMMAND_CUSTOM_BASE

Identifier name	Purpose
VCFNODE_COMMAND_CHANGE_PARAM_VALUE	Change a parameter value. (1) Any node parameter value may be changed after loading a graph (using VCFAPI:Load/LoadFile) but before calling VCFAPI:Run. (2) After calling VCFAPI:Run only a subset of node parameters may be modified. Refer to "Dynamic parameter control" chapter for parameters supporting dynamic modification. Associated data structure: VCFNodeCommandChangeParam
VCFNODE_COMMAND_GET_PARAM_VALUE	Get a parameter value. Note: Current limitation prevents getting parameter value until after node has initialized (after invocation of VCFAPI:Run) Associated data structure: VCFNodeCommandGetParam
VCFNODE_COMMAND_DISABLE	Not supported!
VCFNODE_COMMAND_ENABLE	Not supported!
VCFNODE_COMMAND_CUSTOM_BASE	<Base for custom node commands>

VCFGraphExecutionCompleteStatus

```
typedef enum {
    VCFGRAPH_EXECCOMPLETE_STATUS_STOPPED    = 0,
    VCFGRAPH_EXECCOMPLETE_STATUS_ABORTED    = 1,
    VCFGRAPH_EXECCOMPLETE_STATUS_COMPLETED  = 2,
    VCFGRAPH_EXECCOMPLETE_STATUS_TIMEOUT    = 3
} VCFGraphExecutionCompleteStatus;
```

Identifier name	Purpose
VCFGRAPH_EXECCOMPLETE_STATUS_STOPPED	Execution stopped
VCFGRAPH_EXECCOMPLETE_STATUS_ABORTED	Execution aborted due to an error
VCFGRAPH_EXECCOMPLETE_STATUS_COMPLETED	Execution completed
VCFGRAPH_EXECCOMPLETE_STATUS_TIMEOUT	Execution stopped due to timeout

VCFAPICallbackDataSystemConfig

Only applies to C language binding.

```
typedef struct {
    const char* component;
    const char* name;
    const char* value;
    const char* attribute;
} VCFAPICallbackDataSystemConfig;
```

VCFAPICallbackDataStatus

Only applies to C language binding.

```
typedef struct  {
    const char* status;
    int   res;
} VCFGraphCallbackDataStatus;
```

VCFGraphCallbackDataStatus

Only applies to C language binding.

```
typedef struct  {
    const char* component;
    const char* status;
    int   res;
} VCFGraphCallbackDataStatus;
```

VCFGraphCallbackDataStatusProgress

Only applies to C language binding.

```
typedef struct  {
    const char*     sourceComponent;
    unsigned int    numPackets;
} VCFGraphCallbackDataStatusProgress;
```

VCFGraphCallbackDataExecutionComplete

Only applies to C language binding.

```
typedef struct  {
    VCFGraphExecutionCompleteStatus status;
} VCFGraphCallbackDataExecutionComplete;
```

VCFGraphCallbackDataBenchmark

Only applies to C language binding.

```
typedef struct  {
    const char* data;
} VCFGraphCallbackDataBenchmark;
```

VCFNodeEventContext

VCFNodeEventContext provides details about the origin of the event.

```
struct _VCFNodeEventContext {
```

```
    void*         pGraph;
    const char*   instanceName;
    unsigned int  eventId;
} VCFNodeEventContext;
```

Identifier name	Purpose
pGraph	Originating VCFGraph handle
instanceName	Name of originating node instance in graph
eventId	ID of event that caused the callback

VCFNodeCallbackDataExecutionComplete

```
typedef struct {
    unsigned int numPackets;
} VCFNodeCallbackDataExecutionComplete;
```

Identifier name	Purpose
numPackets	Number of data packets processed

VCFNodeCallbackDataActualConfig

```
typedef struct {
    const char* config;
} VCFNodeCallbackDataActualConfig;
```

Identifier name	Purpose
config	Actual node configuration after completed node initialization. All node parameters are presented as a serialized key/value pair string.

VCFNodeCommandChangeParam

```
typedef struct {
    char* pParamName;
    char* pParamValue;
} VCFNodeCommandChangeParam;
```

Identifier name	Purpose
pParamName	Parameter name
pParamValue	Parameter value

VCFNodeCommandGetParam

```
typedef struct _VCFNodeCommandGetParam {
    char* pParamName;
    char* pParamValue;
```

```
} VCFNodeCommandGetParam;
```

Identifier name	Purpose
pParamName	Parameter name
pParamValue	Parameter value

VCFAPIConfigItemAndroidEnv

```
typedef struct VCFAPIConfigItemAndroidEnv {
    VCFAPIConfigItemType     type;
    void*                    JavaVM;
    const char*              nativeLibraryDir;
} VCFNodeCommandGetParam;
```

Identifier name	Purpose
type	Type of config – use VCFAPIConfigItemTypeAndroidEnv
JavaVM	Handle to Java Virtual Machine
nativeLibraryDir	Absolute path to Native Libraries for the App

Data Structure Reference – Node Specific

This section describe data structures used by specific nodes for commands and events.

VCFEncodeComponentCommands

```
typedef enum {
    VCFNODE_COMMAND_ENCODE_PAYLOAD = VCFNODE_COMMAND_CUSTOM_BASE + 1,
    VCFNODE_COMMAND_ENCODE_INSERT_IDR = VCFNODE_COMMAND_CUSTOM_BASE + 2
} VCFEncodeComponentCommands;
```

Identifier name	Purpose
VCFNODE_COMMAND_ENCODE_PAYLOAD	Request insertion of provided payload into stream. Associated data structure: VCFNodeEncodeCommandPayload
VCFNODE_COMMAND_ENCODE_INSERT_IDR	Request insertion of IDR (key frame) into stream. Associated data structure: N/A

VCFRenderComponentCommands

```
typedef enum {
    VCFNODE_COMMAND_RENDER_SET_TARGET = VCFNODE_COMMAND_CUSTOM_BASE + 1
} VCFRenderComponentCommands;
```

Identifier name	Purpose
VCFNODE_COMMAND_RENDER_SET_TARGET	Configure Render component with render target details. Associated data structure: VCFNodeRenderCommandSetTarget

VCFCameraComponentCommands

```
typedef enum {
       VCFNODE_COMMAND_CAMERA_SET_PREVIEW_SURFACE = VCFNODE_COMMAND_CUSTOM_BASE + 1,
       VCFNODE_COMMAND_CAMERA_SET_ZOOM             = VCFNODE_COMMAND_CUSTOM_BASE + 2
} VCFCameraComponentCommands;
```

Identifier name	Purpose
VCFNODE_COMMAND_CAMERA_SET_PREVIEW_SURFACE	Set preview camera surface holder handle. Note: this command is only applicable to Android Camera node. Associated data structure: VCFNodeCameraCommandSetPreviewSurface

VCFNODE_COMMAND_CAMERA_SET_ZOOM	Set preview camera surface holder handle. Note: this command is only applicable to Android Camera node. Associated data structure: VCFNodeCameraCommandSetZoom

VCFNodeEncodeCommandPayload

```
typedef struct {
    unsigned int frameNo;
    unsigned int type;
    unsigned int dataLengthBits;
    unsigned int dataLengthBytes;
    unsigned char* pData;
} VCFNodeEncodeCommandPayload;
```

Identifier name	Purpose
frameNo	
type	
dataLengthBits	
dataLengthBytes	
pData	

VCFNodeRenderCommandSetTarget

```
typedef struct {
    void* windowsHandle;
    int x;
    int y;
    int w;
    int h;
} VCFNodeRenderCommandSetTarget;
```

Identifier name	Purpose
windowsHandle	
X	
Y	
W	
H	

VCFNodeCameraCommandSetPreviewSurface

```
typedef struct {
       void *androidSurfaceHolder;
} VCFNodeCameraCommandSetPreviewSurface;
```

Identifier name	Purpose

androidSurfaceHolder	

VCFNodeCameraCommandSetZoom

```
typedef struct {
       double zoomRatio;
} VCFNodeCameraCommandSetZoom;
```

Identifier name	Purpose
zoomRatio	

VCFEncodeComponentEvents

```
typedef enum {
    VCFNODE_EVENT_ENCODE_STREAM_INFO        = (VCFNODE_EVENT_CUSTOM_BASE * 0x01)
} VCFEncodeComponentEvents;
```

Identifier name	Purpose
VCFNODE_EVENT_ENCODE_STREAM_INFO	Subscribe to encoded stream info details. Includes information such as frame type size and frame size. Associated data structure: VCFNodeEncodeCallbackDataStreamInfo

VCFDecodeComponentEvents

```
typedef enum {
    VCFNODE_EVENT_DECODE_PAYLOAD                = (VCFNODE_EVENT_CUSTOM_BASE * 0x01),
    VCFNODE_EVENT_DECODE_STREAM_HEADER_CHANGED  = (VCFNODE_EVENT_CUSTOM_BASE * 0x02)
} VCFDecodeComponentEvents;
```

Identifier name	Purpose
VCFNODE_EVENT_DECODE_PAYLOAD	Subscribe to decoded payload data. Associated data structure: VCFNodeDecodeCallbackDataPayload
VCFNODE_EVENT_DECODE_STREAM_HEADER_CHANGED	Subscribe to event indicating that stream header has changed. Associated data structure: VCFNodeDecodeCallbackDataStreamHeaderChanged

VCFNodeEncodeCallbackDataStreamInfo

```
typedef struct {
    unsigned int frameSize;
    unsigned short frameType;
```

```
} VCFFrameSizeContainer;

typedef struct {
    VCFFrameSizeContainer*  pFrameSizeInfo;
    unsigned int            numFrameSizeItems;
    unsigned int            capturedRangeStart;
    unsigned int            capturedRangeStop;
    unsigned int            totalSize[8]; // I=1, P=2, B=4
    unsigned int            numFrames[8]; // I=1, P=2, B=4
    unsigned int            maxFrameSize[8]; // I=1, P=2, B=4
    unsigned int            minFrameSize[8]; // I=1, P=2, B=4
    unsigned int            FrameRateExtN;
    unsigned int            FrameRateExtD;
} VCFNodeEncodeCallbackDataStreamInfo;
```

Identifier name	Purpose
frameSize	
frameType	
pFrameSizeInfo	
numFrameSizeItems	
capturedRangeStart	
capturedRangeStop	
totalSize	
numFrames	
maxFrameSize	
minFrameSize	
FrameRateExtN	
FrameRateExtD	

VCFNodeDecodeCallbackDataStreamHeaderChanged

```
typedef struct {
    int frameNo;
} VCFNodeDecodeCallbackDataStreamHeaderChanged;
```

Identifier name	Purpose
frameNo	

VCFNodeDecodeCallbackDataStreamPayload

```
typedef struct {
    unsigned int frameNo;
    unsigned long long timeStamp;
    unsigned int type;
    unsigned int dataLengthBits;
    unsigned char* pData;
} VCFNodeDecodeCallbackDataPayload;
```

Identifier name	Purpose
frameNo	

timeStamp	
type	
dataLengthBits	
pData	

Dynamic Parameter Reference – Node Specific

This section describe the node parameters supporting dynamic modification of value during graph execution.

Node Type	Parameter name	Description
VCFVideoEncode	bitRate	Control encoder bit rate. Note that the rate must stay within the range of max bit rate specified at encoder initialization.
VCFVideoEncode	QPI	Control Quantization Parameter for Constant Quantization Parameter (CQP) rate control mode.
VCFVideoProcess	Brightness	Frame processing brightness control. Member of the “Amplifiers” property group.
VCFVideoProcess	Contrast	Frame processing contrast control. Member of the “Amplifiers” property group.
VCFVideoProcess	Hue	Frame processing hue control. Member of the “Amplifiers” property group.
VCFVideoProcess	Saturation	Frame processing saturation control. Member of the “Amplifiers” property group.
VCFVideoProcess	DetailFactor	Frame detail filter control. Member of the “Detail” property group.
VCFVideoProcess	DenoiseFactor	Frame denoising control. Member of the “Denoise” property group.
VCFVideoProcess	cropWidth	Frame width cropping control. Cropping properties are enabled by enabling the “Basic:General:Cropping” property
VCFVideoProcess	cropHeight	Frame height cropping control. Cropping properties are enabled by enabling the “Basic:General:Cropping” property
VCFVideoProcess	cropX	Frame x-offset cropping control. Cropping properties are enabled by enabling the

		“Basic:General:Cropping” property
VCFVideoProcess	cropY	Frame y-offset cropping control. Cropping properties are enabled by enabling the “Basic:General:Cropping” property