GoForce

Drivers
API Reference Manual for SC15

DA-2045-001v02
February 6, 2006

Published by
NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

Notice

Trademarks

Copyright

Revision History

This manual is targeted for the SC15 Media Processor.

Current Changes

This is the preliminary release of the GoForce Drivers API Reference Manual for SC15.

Date	Change
1/10/06	Initial Release, with updates to GFIx, GFRmIx,GFI2C, GFJxDec, GFMxDec, GFMXDecH264, and GFMxEncH264 APIs.
2/6/06	Added "Resource Manager Read DMA (GFRmRDMA)" on page 579.

Table of Contents

CHAPTER

1

GoForce Software Development Kit (GFSDK)

Introduction

The NVIDIA® GoForce™ software development kit (GFSDK) consists of the GoForce application programming interface (API) and sample applications that show how to use the API. The GoForce API contains component APIs and services. Each component API is a thin software abstraction layer that represents hardware functionality. The purpose of the GFSDK is to ease the integration of NVIDIA's GoForce family of media processors into various real time operating systems (RTOS) and platforms. Most of the GFSDK code is agnostic about operating systems and platforms and is written in ANSI C that can be compiled for many processors, even those with primitive compilers.

NVIDIA's driver packages for popular embedded operating systems—such as Windows CE™, Palm OS™ and Symbian™—accelerate the standard operating system API as much as possible with various hardware features. On some platforms where the standard operating system API does not support certain advanced hardware features, a subset of the GFSDK functionality is available.

The GoForce API is not intended to compete against standard operating system APIs or against other standard APIs, such as OpenGL ES; it is intended to complement those APIs by supporting new hardware features.

In a typical system design, the GoForce companion processor is connected to the system (memory) bus of a baseband CPU or application-processor CPU as shown in Figure 1.

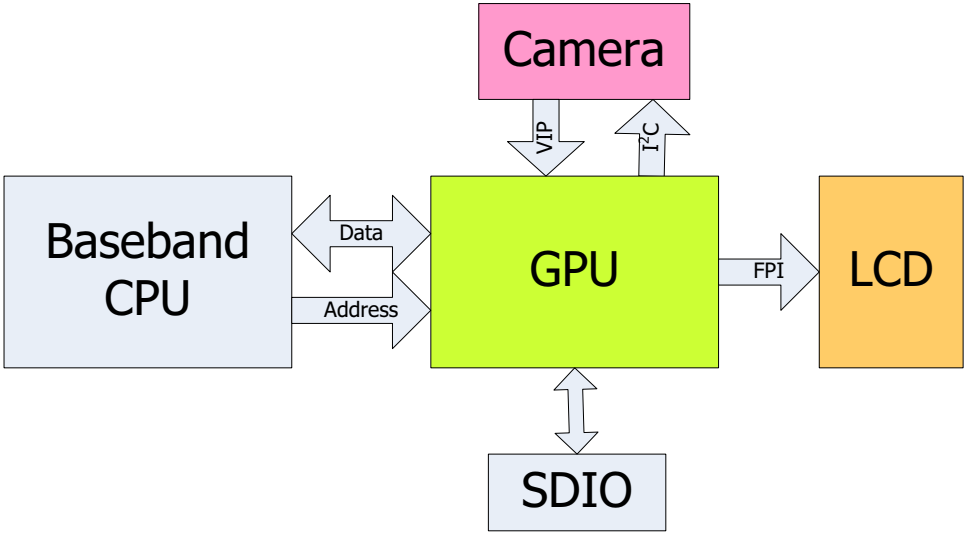


Figure 1. Basic System Topology

In a flip phone where the GoForce processor sits behind a panel, host addressing can be reduced to 1- or 2-bit for indirect mode to minimize both cost and the EMI across flex cable.

A GoForce evaluation board can also be driven by a computer using a PCI adaptor that has a PCI bridge chip to connect the GoForce host interface to the PCI bus as shown in Figure 2.

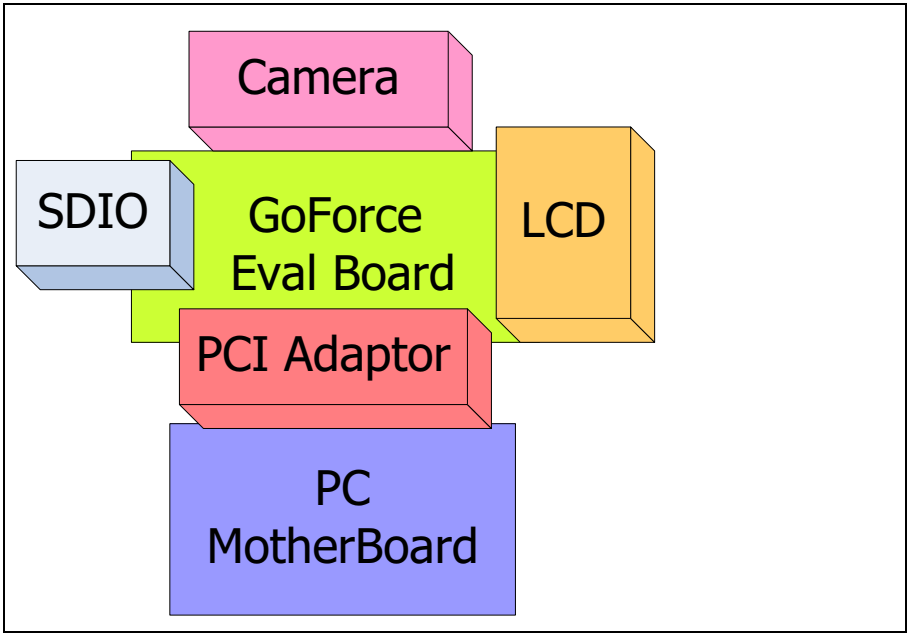


Figure 2. PC PCI System

The following diagrams show the role of the GFSDK in different operating system environments.

- Figure 3., “PCI-Based PC System for Use Cases” on page 4
- Figure 4., “GFSDK Architecture with RTOS” on page 5
- Figure 5., “Software Architecture for Windows CE 5.0” on page 6
- Figure 6., “Software Architecture for Palm OS 6.0” on page 7
- Figure 7., “Software Architecture for Symbian” on page 8

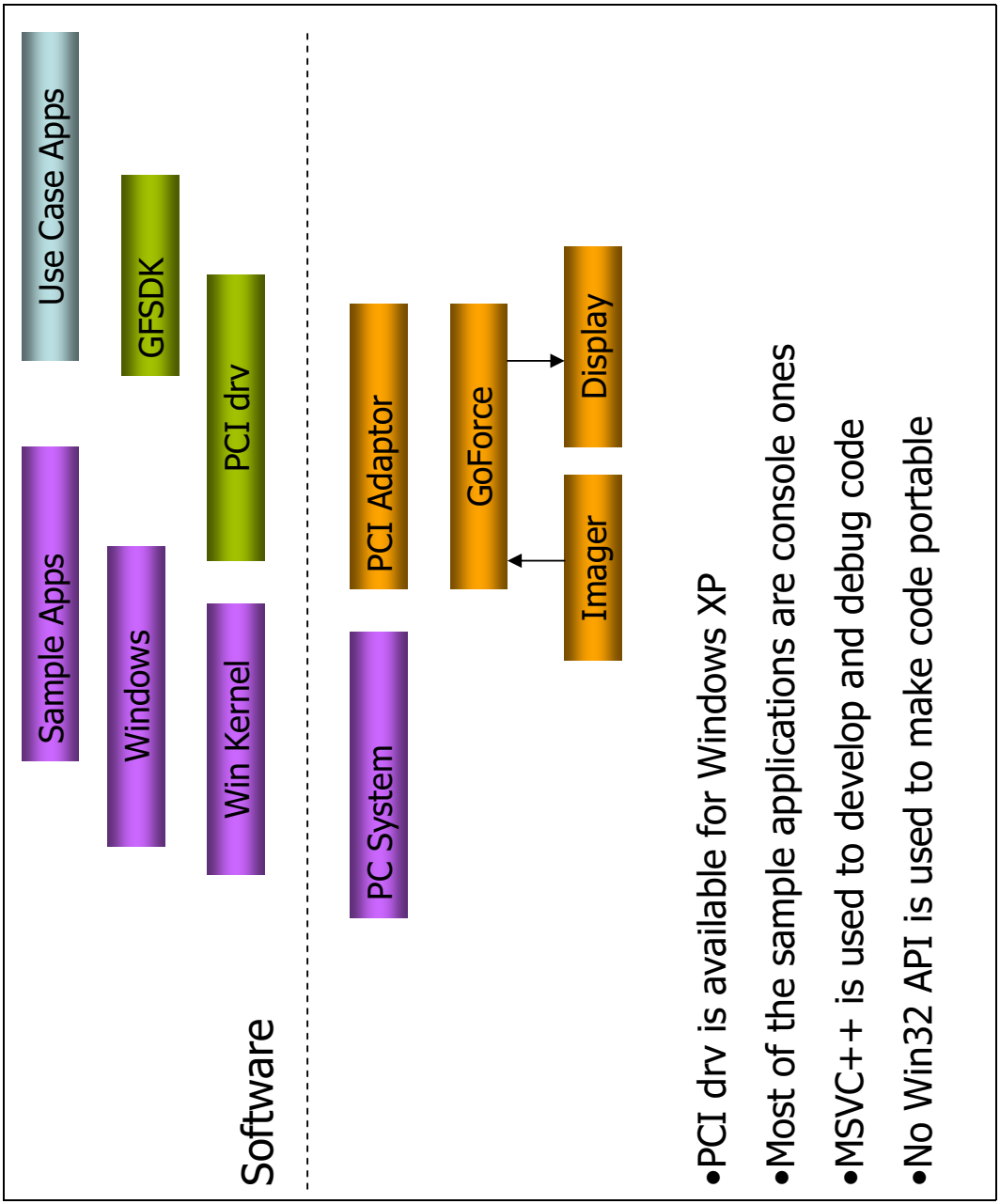


Figure 3. PCI-Based PC System for Use Cases

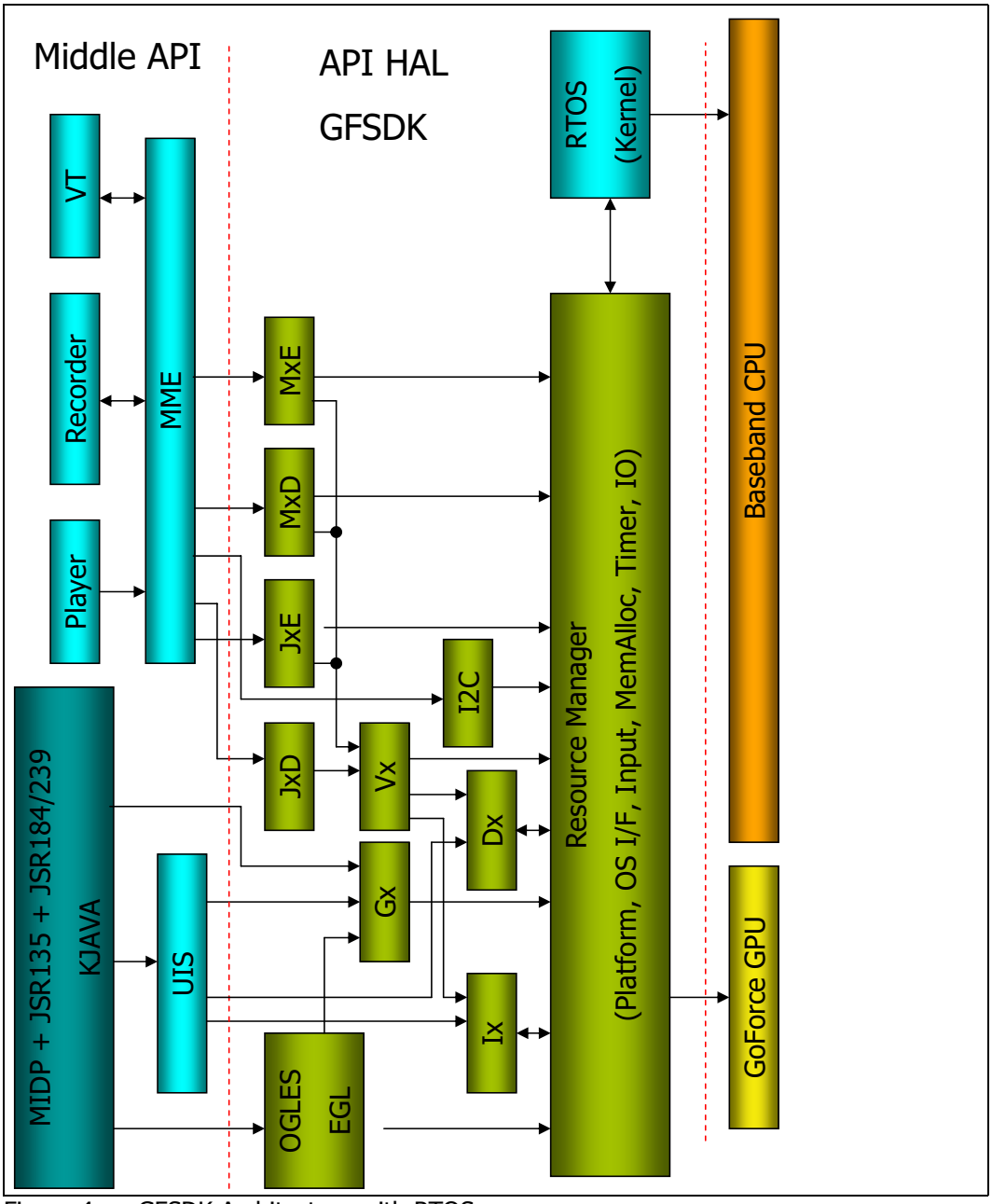


Figure 4. GFSDK Architecture with RTOS

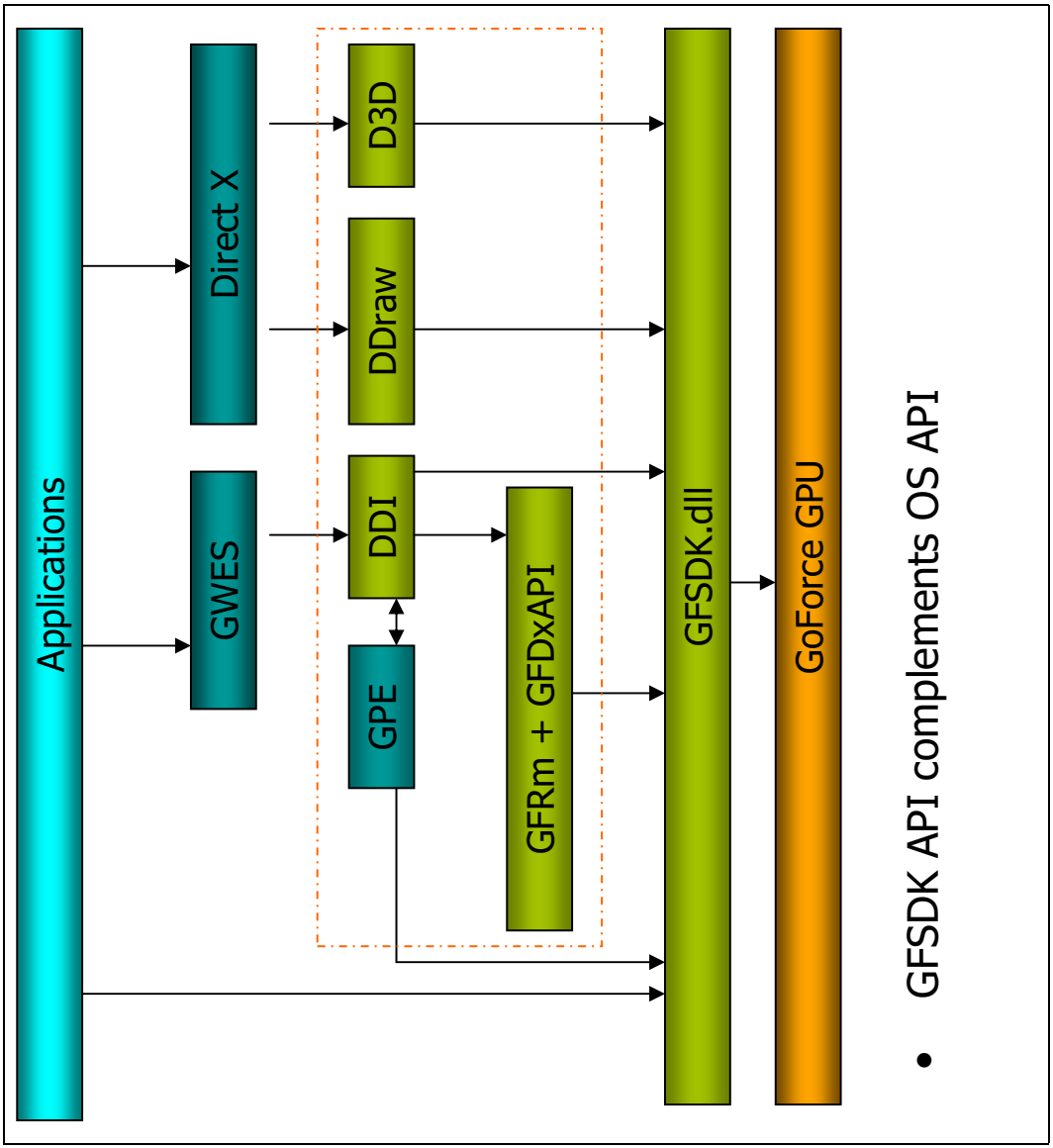


Figure 5. Software Architecture for Windows CE 5.0

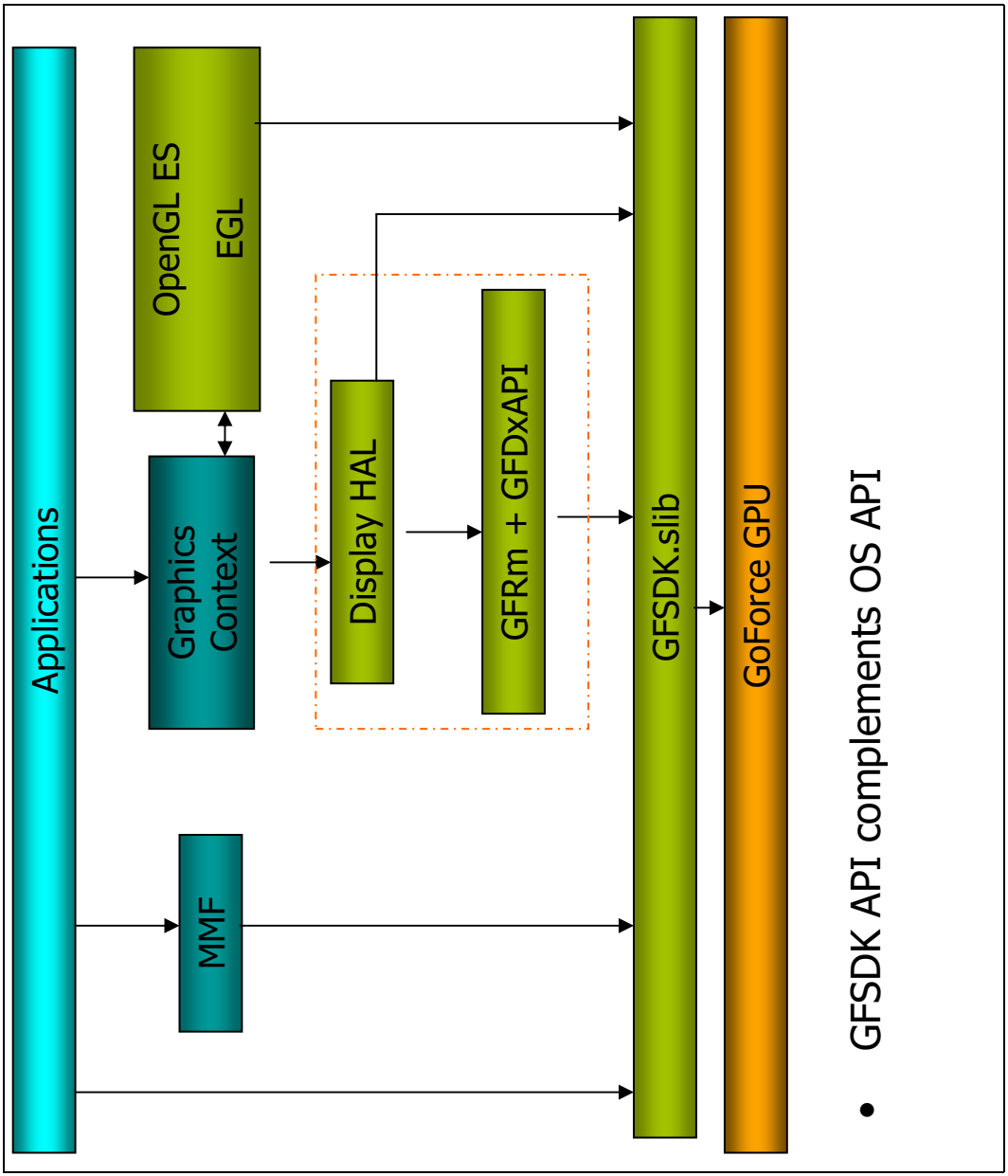


Figure 6. Software Architecture for Palm OS 6.0

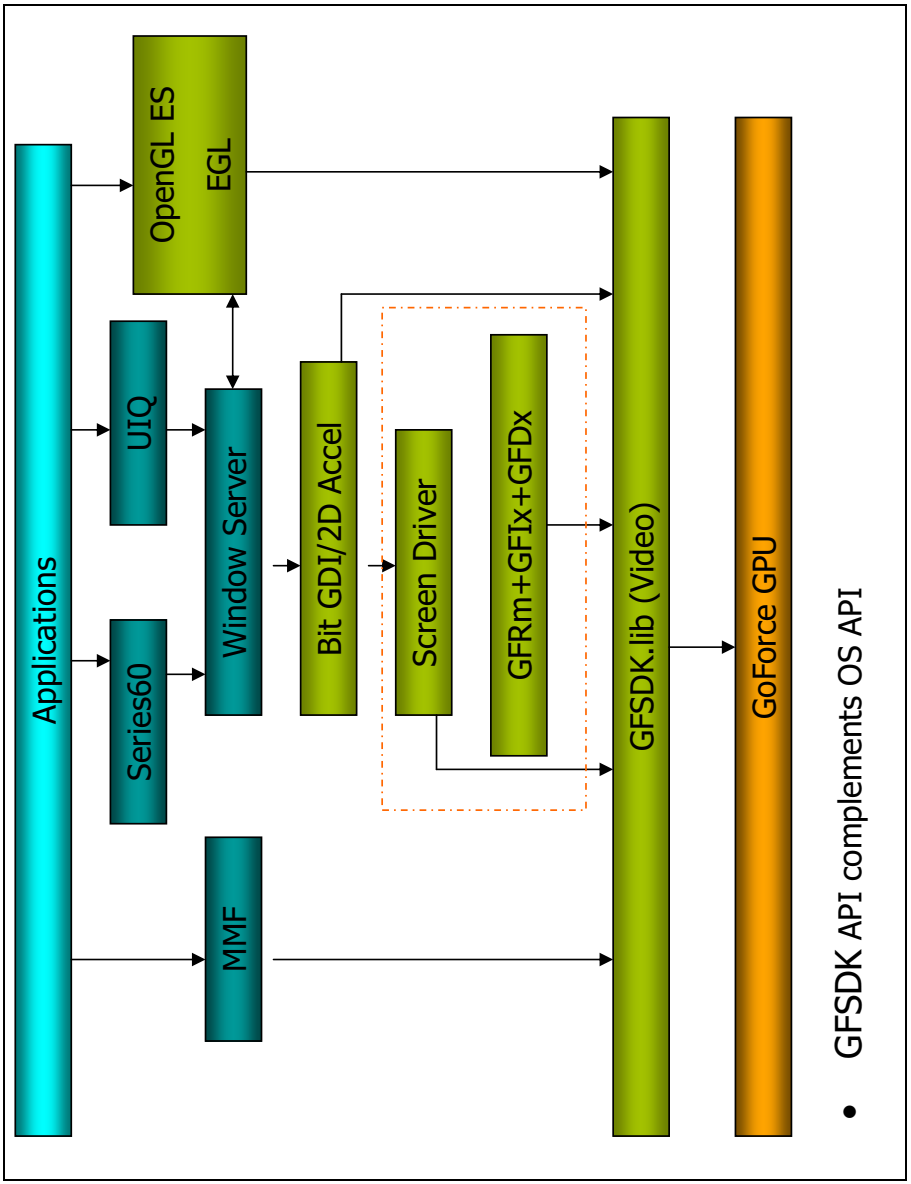


Figure 7. Software Architecture for Symbian

Multithreading and Multiprocess Support

Hardware accesses in most of the GFSDK APIs are serialized by the use of hardware protection semaphores. The GFRm (resource manager) and GFVxAPI (video) provide thread-safe execution. They use the semaphores to access global memory data structures so that multiple threads can use the data structures without corrupting them.

The GFRm, GFIxAPI, and GFDxAPI are low-level APIs whose default function versions—such as **`GFRmSurfaceUpdate()`**—are not serialized with semaphores because these APIs are usually called from semaphore-protected areas of other APIs. If you call these low-level APIs directly from an application, please use the thread-safe function versions: those with the suffix "**`WithCS`**" (with critical sections)—**`GFRmSurfaceUpdateWithCS()`** in this case. All other APIs provide serialized access to the media processor hardware.

Scope of This Manual

The component APIs and services described in this document include the following:

GFSDK Common Data Structures and Types

These data structures and types are used by the individual GoForce APIs that are included in this document:

GFSDK Data Types

This table lists the common data types that are abstracted in the GFSDK to enhance portability, future migration, and ease of use.

GFSDK Data Types

`NvU32`	32-bit unsigned integer.
`NvS32`	32-bit signed integer.
`NvU16`	16-bit unsigned integer.
`NvS16`	16-bit signed integer.
`NvU8`	8-bit unsigned integer.
`NvS8`	8-bit signed integer.
`void *`	Pointer to `void`.
`GF_HANDLE`	Handle specific to GFSDK.

GFPROPERTY

`GFPROPERTY` is the common structure that describes the generic properties of the NVIDIA media processor. Individual APIs may have definitions for **`Capability`** that are unique to them.

GFPROPERTY Structure

```
typedef struct _GFPROPERTY{
    NvU32  Version;
```

GFPROPERTY Structure (continued)

```
    NvU16  DeviceID;
    NvU16  DeviceRev;
    NvU32  BuildNumber;
    NvU32  Capability;
    NvU32  CapabilityEx;
    NvU32  ProdSkuID;
} GFPROPERTY, *PGFPROPERTY;
```

GFPROPERTY Fields

`Version`	GFSDK component version number.
`DeviceID`	Device ID (`0x2200` for GoForce 3000).
`DeviceRev`	Silicon revision supported.
`BuildNumber`	Build number.
`Capability`	Individual component's capability.
`CapabilityEx`	Individual component's extended capability.
`ProdSkuID`	EFUSE Product SKU ID

GFRECT

`GFRECT` describes a rectangle. NVIDIA media processors always work with all four borders of a rectangle inclusively. The width of a rectangle described by **`GFRECT`** is (**`right`** - **`left`** + 1), and the height is (**`bottom`** - **`top`** + 1).

GFRECT Structure

```
typedef struct  _GFRECT {
    NvU16  left;
    NvU16  top;
    NvU16  right;
    NvU16  bottom;
} GFRECT, *PGFRECT;
```

GFRECT Fields

`left`	The rectangle's left position horizontally.
`top`	The rectangle's top position vertically.
`right`	The rectangle's right position horizontally.
`bottom`	The rectangle's bottom position vertically.

GF_RETTYPE Status and Error Codes

GF_RETTYPE, a 32-bit signed value, is the common data type that provides status and error codes for all services and API functions. When an error condition is returned, the most significant bit (MSB) is set to indicate a negative value. Each GFSDK component API or service has its own specific status values that are specified with each individual service or function.

GF_RETTYPE Definitions

```
/*
  Two most common default status values
*/
#define GF_SUCCESS                              0x00000000L
    // Call successful.
#define GF_ERROR                                0x80000000L
    //  Call failed.
/*
    Other common default status values
*/
#define GF_SUCCESS_ADDITIONAL                   0x00000001L
    // Additional success status provided.
#define GF_ERROR_ADDITIONAL                     0x80000001L
    //  Additional error status provided.
#define GF_ERROR_NO_SUPPORT                     0x80000002L
    //  No support for this call.
#define GF_ERROR_OUT_MEMORY                     0x80000003L
    //  Out-of-memory error condition.
#define GF_ERROR_NO_PRIMARY_SURFACE             0x80000004L
    //  No primary surface available.
#define GF_ERROR_BAD_PARAMETER                  0x80000005L
    //  Bad parameter when calling API or service.
#define GF_ERROR_DEVICE_UNINITIALIZED           0x80000006L
    //  Device is not initialized.
#define GF_COMPONENT_ERROR_SHIFT       24
#define GF_COMPONENT_REVERSE(c)      ((((c)&64)>>6)|
(((c)&32)>>4)|(((c)&16)>>2)|((c)&8)|(((c)&4)<<2)|(((c)&2)<<4)|
(((c)&1)<<6))

#define EPP_ERROR
(GF_COMPONENT_REVERSE(GF_EPPAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define GX_ERROR
(GF_COMPONENT_REVERSE(GF_GXAPI)<<GF_COMPONENT_ERROR_SHIFT)
```

GF_RETTYPE Definitions (continued)

```
#define VX_ERROR
(GF_COMPONENT_REVERSE(GF_VXAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define JX_ERROR
(GF_COMPONENT_REVERSE(GF_JXEAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define JXD_ERROR
(GF_COMPONENT_REVERSE(GF_JXDAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define MXE_ERROR
(GF_COMPONENT_REVERSE(GF_MXEAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define MXD_ERROR
(GF_COMPONENT_REVERSE(GF_MXDAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define CPU_ERROR
(GF_COMPONENT_REVERSE(GF_CPUAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define DX_ERROR
(GF_COMPONENT_REVERSE(GF_DXAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define IX_ERROR
(GF_COMPONENT_REVERSE(GF_IXAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define ISP_ERROR
(GF_COMPONENT_REVERSE(GF_ISPAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define FDEV_ERROR
(GF_COMPONENT_REVERSE(GF_FDEVAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define BDEVSD_ERROR
(GF_COMPONENT_REVERSE(GF_BDEVSDAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define BDEVMD_ERROR
(GF_COMPONENT_REVERSE(GF_BDEVMDAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define UI_ERROR
(GF_COMPONENT_REVERSE(GF_UIAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define OSX_ERROR
(GF_COMPONENT_REVERSE(GF_OSXAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define I2C_ERROR
(GF_COMPONENT_REVERSE(GF_I2CAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define I2S_ERROR
(GF_COMPONENT_REVERSE(GF_I2SAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define MMPROC_ERROR
(GF_COMPONENT_REVERSE(GF_MMPROCAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define CAM_ERROR
(GF_COMPONENT_REVERSE(GF_CAMAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define G3D_ERROR
(GF_COMPONENT_REVERSE(GF_3DAPI)<<GF_COMPONENT_ERROR_SHIFT)
#define INTX_ERROR
(GF_COMPONENT_REVERSE(GF_INTXAPI)<<GF_COMPONENT_ERROR_SHIFT)
```

GF_RETTYPE Definitions (continued)

```
#define MXDH264_ERROR
(GF_COMPONENT_REVERSE(GF_MXDH264API)<<GF_COMPONENT_ERROR_SHIFT
)
#define MXEH264_ERROR
(GF_COMPONENT_REVERSE(GF_MXEH264API)<<GF_COMPONENT_ERROR_SHIFT
)
#define MXDHVC1_ERROR
(GF_COMPONENT_REVERSE(GF_MXDVC1API)<<GF_COMPONENT_ERROR_SHIFT)
#define RM_ERROR
(GF_COMPONENT_REVERSE(GF_RMAPI)<<GF_COMPONENT_ERROR_SHIFT)

#define GFEPP_ERROR           (GF_ERROR | EPP_ERROR)
#define GFGX_ERROR                    (GF_ERROR | GX_ERROR)
#define GFVX_ERROR                    (GF_ERROR | VX_ERROR)
#define GFJX_ERROR                    (GF_ERROR | JX_ERROR)
#define GFJXD_ERROR                   (GF_ERROR | JXD_ERROR)
#define GFMXE_ERROR                   (GF_ERROR | MXE_ERROR)
#define GFMXD_ERROR                  (GF_ERROR | MXD_ERROR)
#define GFCPU_ERROR                   (GF_ERROR | CPU_ERROR)
#define GFDX_ERROR                    (GF_ERROR | DX_ERROR)
#define GFIX_ERROR                    (GF_ERROR | IX_ERROR)
#define GFISP_ERROR                   (GF_ERROR | ISP_ERROR)
#define GFFDEV_ERROR                  (GF_ERROR | FDEV_ERROR)
#define GFBDEVSD_ERROR                (GF_ERROR | BDEVSD_ERROR)
#define GFBDEVMD_ERROR                (GF_ERROR | BDEVMD_ERROR)
#define GFUI_ERROR                    (GF_ERROR | UI_ERROR)
#define GFOSX_ERROR                   (GF_ERROR | OSX_ERROR)
#define GFI2C_ERROR                   (GF_ERROR | I2C_ERROR)
#define GFI2S_ERROR                   (GF_ERROR | I2S_ERROR)
#define GFMMPROC_ERROR                (GF_ERROR | MMPROC_ERROR
#define GFCAM_ERROR                   (GF_ERROR | CAM_ERROR)
#define GF3D_ERROR                    (GF_ERROR | G3D_ERROR)
#define GFINTX_ERROR                  (GF_ERROR | INTX_ERROR)
#define GFMXDH264_ERROR            (GF_ERROR | MXDH264_ERROR)
#define GFMXEH264_ERROR               (GF_ERROR | MXEH264_ERROR)
#define GFMXDVC1_ERROR                (GF_ERROR | MXDHVC1_ERROR)
#define GFRM_ERROR                     (GF_ERROR | RM_ERROR)
```

Two helpful C macros have been provided to indicate general success or error conditions:

GF_RETTYPE Macro Definitions

```
#define ISGFSUCCESS( code )          ((code) >= GF_SUCCESS)
#define ISGFERROR( code )            ((code) < GF_SUCCESS)
```

GF_INTERRUPT_STATUS_TYPE

These are the states of a given interrupt bit. The information indicates if the status of the intended interrupt bit is `TRUE` or `FALSE`, and if the bit has been enabled so that its status is reflected in the main interrupt signal. Status is constantly given for most hardware interrupt bits to indicate their current state. However, even if the interrupt bit status is `TRUE`, it does not cause a real interrupt unless it has been enabled.

GF_INTERRUPT_STATUS_TYPE Enumerated Type

```
typedef enum{
    GF_INTERRUPT_OFF_STATUS_FALSE,
        // Component-level interrupt is disabled.
        // The interrupt status is false.
    GF_INTERRUPT_ON_STATUS_FALSE,
        // Component-level interrupt is enabled.
        // The interrupt status is false.
    GF_INTERRUPT_OFF_STATUS_TRUE,
        // Component-level interrupt is disabled.
        // The interrupt status is true.
    GF_INTERRUPT_ON_STATUS_TRUE
        // Component-level interrupt is enabled.
        // The interrupt status is true.
}  GF_INTERRUPT_STATUS_TYPE;
```

GFGPIOSTATUS

This structure returns GPIO pin status and holds common GPIO register information that includes input-data, input-enable, and output-enable status bits.

The bits can be set to `1` or cleared to `0`, and their status can be obtained.

GFGPIOSTATUS Structure

```
typedef struct _GFGPIOSTATUS
{
    NvU32       in_data;      /**< Input data bit */
    NvU32       in_enable;    /**< Input enable bit */
    NvU32       out_enable;   /**< Output enable bit */
    NvU32       out_data;     /**< Output data bit */
    NvU32       out_select;   /**< Output select bit */
} GFGPIOSTATUS, *PGFGPIOSTATUS;
```

CHAPTER 2

Resource Manager Services (GFRm)

Resource Manager Services

The resource manager services (GFRm) comprise the following groups of services:

Object Manager Services

The object manager services are defined by the functions, attributes, and states that follow in the next sections.

Object Manager Functions

The object manager functions include the following:

GFRmOpen()

This function initializes the resource manager. An application should call this function before calling any other GFSDK service. If an error is returned by this function, no GFSDK service is supported.

Function Prototype

```
GF_OPEN_RETTYPE  GFRmOpen(
    GF_OPEN_HANDLE  RmOpenHandle );
```

Parameters

`RmOpenHandle`	Operating system dependent. (Set to NULL for now.)

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmClose()

This function closes the resource manager. All necessary cleaning up is performed here. The application is required to call this function. The **pRmHandle** is set to NULL if the call is successful.

Function Prototype

```
GF_RETTYPE  GFRmClose(
    GF_HANDLE  *pRmHandle );
```

Parameters

*pRmHandle	Pointer to handle specific to the GFRm.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmGetProperty()

Returns the properties for this instance of the resource manager. Check the capability flags to determine what is available.

Function Prototype

```
GF_RETTYPE  GFRmGetProperty(
    GF_HANDLE   RmHandle,
    GFPROPERTY  *pRmProp );
```

Parameters

RmHandle	Handle specific to the GFRm.
*pRmProp	Pointer to "GFPROPERTY Structure" on page 10.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmSetAttribute()

This is a generic attribute setting routine. See "Object Manager Attributes" on page 21.

Function Prototype

```
GF_RETTYPE  GFRmSetAttribute(
    GF_HANDLE          RmHandle,
    GF_HANDLE          ObjectHandle,
    GF_ATTRIBUTE_TYPE  AttribType,
    NvU32              AttribData );
```

Parameters

`RmHandle`	Handle specific to the GFRm.
`ObjectHandle`	Pointer to object handle. (*Not used.*)
`AttribType`	Attribute type.
`AttribData`	Attribute data.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR_BAD_PARAMETER`	If parameter is bad or unrecognizable.
`GF_ERROR`	If other error.

GFRmGetAttribute()

This is a generic attribute getting routine. See "Object Manager Attributes" on page 21.

Function Prototype

```
GF_RETTYPE  GFRmGetAttribute(
    GF_HANDLE          RmHandle,
    GF_HANDLE          ObjectHandle,
    GF_ATTRIBUTE_TYPE  AttribType,
    NvU32              *pAttribData );
```

Parameters

`RmHandle`	Handle specific to the GFRm.
`ObjectHandle`	Pointer to object handle. (*Not used.*)

Parameters (continued)

`AttribType`	Attribute type.
`*pAttribData`	Pointer to attribute data.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If an error other than `GF_ERROR_NO_SUPPORT`.
`GF_ERROR_NO_SUPPORT`	If the specified `AttribType` is not supported by this service.

Object Manager Attributes

These are the attributes used by the object manager.

Object Manager Attributes Enumeration Type

```
typedef enum {
    GF_ATTRIB_DEVICE_ADDRESS,
    GF_ATTRIB_IO_MAP_BASE,
        // Mapped/virtual/usable base address for register
        // I/O space.
    GF_ATTRIB_EMBEDDED_MEM_MAP_BASE,
        // Mapped/virtual/usable base address for embedded
        // memory space.
    GF_ATTRIB_INIT_EMBEDDED_MEM_TOTAL,
        // Initial usable embedded memory total size.
        // Get this attribute after Dx initializes LCD.
    GF_ATTRIB_ASSOCIATE,          // User-specific objects
    GF_ATTRIB_USER_OBJECT_0,
    GF_ATTRIB_USER_OBJECT_1,
    GF_ATTRIB_USER_OBJECT_2,
    GF_ATTRIB_USER_OBJECT_3,
    GF_ATTRIB_SWAP_SHARE_MEMORY_ALLOC_ON_MAX_CHUNK,
        // Swap memory allocation with shared embedded memory
        // scheme on MAX CHUNK allocation.
    GF_ATTRIB_DEVICE_INFO,
        // Run-time info about device: 31:16 = rev, 15:0 = ID.
    GF_ATTRIB_SURFACE_ABSOLUTE_ROTATE,
        // Set/clear surface with absolute rotation option.
        // Default is to not have absolute rotation.
```

Object Manager Attributes Enumeration Type (continued)

```
    GF_ATTRIB_HW_RES_CONSTRAINT,
        // Enable/disable hardware resource constraint
        // checking. Default is to disable.
    GF_ATTRIB_SURFACE_ABSOLUTE_ROTATE
        // Set/clear surface with absolute rotation option.
} GF_ATTRIBUTE_TYPE;
```

Object Manager States

Object manager states are used to indicate generic and general purpose settings in a particular GFRm service.

Object Manager State Definitions

```
#define GF_STATE_REGISTER                    0x00000001
#define GF_STATE_UNREGISTER                  0x00000002
#define GF_STATE_UNREGISTER_ALL              0x00000004
    // INTERNAL USE ONLY.
#define GF_STATE_ENABLE                      0x00000008
#define GF_STATE_DISABLE                     0x00000010
#define GF_STATE_DONE                        0x00000020
#define GF_STATE_NEW_OR_SHARE                0x00000100
    // Share if possible, else create new.
#define GF_STATE_NEW_ONLY                    0x00000200
    // Only create new.
#define GF_STATE_SHARE_ONLY                  0x00000400
    // Always share.
#define GF_STATE_BLOCK_DEVICE                0x80000000
    // Special Block Device (For internal use only).
#define GF_STATE_DEFAULT          GF_STATE_NEW_OR_SHARE
#define GF_STATE_MASK                        0x000000FF
    // Masks for the regular states.
```

Context Manager Services

The component manager functions include the following:

- "GFRmContextGet()"
- "GFRmContextRelease()"
- "GFRmContextId()"

GFRmContextGet()

This function retrieves either the default context or a new context.

Function Prototype

```
GF_RETTYPE
GFRmContextGet( GF_HANDLE RmHandle,
                NvU32 flags,
                GF_HANDLE *pContext);
```

Parameters

RmHandle	Handle to the resource manager.
flags	#GF_CTX_DEFAULT or #GF_CTX_NEW
*pContext	Pointer to the retrieved context

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmContextRelease()

This function releases a context.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmContextRelease( GF_HANDLE RmHandle,
                    GF_HANDLE *pContext);
```

Parameters

RmHandle	Handle to the resource manager.
*pContext	Pointer to the Context GFRmContextGet

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmContextId()

Each context has an associated ID. This function returns the associated id. id values ranges from 0 to maximum value of contexts available.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmContextId(GF_HANDLE Context,
              NvU32 *id);
```

Parameters

Context	Context handle
*id	Pointer to the context id a NvU32 integer

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Component Manager Services

These services manage the GFSDK components. A component contains APIs that expose the NVIDIA media processor's features.

Component Manager Functions

The component manager functions include the following:

GFRmComponentRegister()

This function is called by the component to register itself with the resource manager. See "Component Enumeration" on page 28.

Function Prototype

```
GF_RETTYPE  GFRmComponentRegister(
    GF_HANDLE       RmHandle,
    GFRMCOMPONENT   *pComponent,
    NvU32           RegisterState );
```

Parameters

`RmHandle`	Handle specific to the GFRm.
`*pComponent`	Pointer to component structure.
`RegisterState`	State of this component.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmComponentGet()

Call this function to obtain a specific component's handle. Use the state argument (see "Object Manager States" on page 22) to specify if a `shared` or `new` component is to be obtained.

Function Prototype

```
GF_RETTYPE  GFRmComponentGet(
    GF_HANDLE         RmHandle,
    GFRMCOMPONENTID   *pComponentID,
    GF_HANDLE         *pComponent,
    GF_STATE_TYPE     state );
```

Parameters

`RmHandle`	Handle specific to the GFRm.
`*pComponentID`	Pointer to component ID structure specifying desired component.
`*pComponent`	Pointer to handle to available component. (NULL if component not available).
`state`	Indicates a newly instantiated or shared component.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmComponentGetEx()

This function gets a registered component, associated with the passed context. Depending on the requested state, a newly instantiated component or a shared component is returned. GF_STATE_DEFAULT is the default option for the state parameter. If the Context handle passed is NULL, then a default context is associated with the component.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmComponentGetEx( GF_HANDLE         RmHandle,
                    GFRMCOMPONENTID   *pComponentID,
                    GF_HANDLE         *pComponent,
                    GF_STATE_TYPE     state,
                    GF_HANDLE         CtxHandle);
```

Parameters

RmHandle	Handle to the resource manager.
*pComponentID	Pointer to component ID struct specifying component.
*pComponent	Pointer to the Handle of the available component. NULL if not available.
state	
CtxHandle	Context Handle. Can be allocated by a call to #GFRmContextGet function.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmComponentRelease()

Releases a component handle. If the component is still being shared by another caller, only the internal reference counter is decremented. When this component is completely released, the component's associated runtime states are released as well. The **pComponent** variable is set to NULL upon a successful release.

Function Prototype

```
GF_RETTYPE  GFRmComponentRelease(
    GF_HANDLE  RmHandle,
    GF_HANDLE  *pComponent )
```

Parameters

RmHandle	Handle specific to the GFRm.
*pComponent	Pointer to component handle to be released.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmComponentEnum()

This function enumerates the registered components in the resource manager.

Function Prototype

```
GF_RETTYPE  GFRmComponentEnum(
    GF_HANDLE        RmHandle,
    GFRMCOMPONENTID  *pComponentId,
    GF_HANDLE        *pEnumHandle,
    GF_STATE_TYPE    state );
```

Parameters

RmHandle	Handle specific to the GFRm.
*pComponentID	Pointer to component type ID.
*pComponent	Pointer to handle for component enumeration.
state	(*Reserved for internal use.*) Set to 0.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Component Enumeration. To begin this service, pass a pointer to a **GF_HANDLE** variable, pre-set to NULL, through the **pEnumHandle** argument, and pass a **pComponentId** argument that points to a **GFRMCOMPONENTID** variable. The service returns a registered component's information in the structure pointed to by **pComponentId** and provides a handle through **pEnumHandle** that can be used for next enumeration call. This service has to be called in a loop until ***pEnumHandle** contains NULL. A sample calling sequence is shown below:

```
GF_HANDLE  EnumHandle;
GFRMCOMPONENTID id;
/*
    Initialize EnumHandle to NULL first
*/
EnumHandle = (GF_HANDLE)NULL;
do
{
    GFRmComponentEnum( RmHandle, &id, &EnumHandle, 0 );
    if ( EnumHandle )
    {
```

```
        // A registered component is obtained, try getting
        // its handle. May want to check id first.
        if ( ISGFSUCCESS( GFRmComponentGet( RmHandle,
        &id, &pComponent, GF_STATE_NEW_OR_SHARE ) ) )
        {
            // Process
        }
    }
}
while( EnumHandle ); // Exits loop when EnumHandle is NULL
```

Component Manager Structures

The **GFRMCOMPONENT** and **GFRMCOMPONENTID** structures are used to identify a component. Because the GFSDK can support similar components (with the same **ComponentType**) for different devices, a device ID and revision are required for component identification. If only one device is to be supported, **GF_DEVICE_ID_DEFAULT** and **GF_DEVICE_REV_DEFAULT** can be used as the arguments in most cases.

GFRMCOMPONENT is an internal structure used by a GFSDK component during the registration process. It is not usually used by an application.

GFRMCOMPONENT Structure

```
typedef struct  _GFRMCOMPONENT {
    GFCOMPONENTID  id;
    GF_RETTYPE     (*Open)(GF_HANDLE);
    GF_RETTYPE     (*Close)(GF_HANDLE);
} GFRMCOMPONENT, *PGFRMCOMPONENT;
```

GFRMCOMPONENT Fields

id	See the "GFRMCOMPONENT Structure" on page 29.
Open	Internal function used during GFRmComponentGet().
Close	Internal function used during GFRmComponentRelease().

GFRMCOMPONENTID Structure

```
typedef struct  _GFRMCOMPONENTID {
    NvU32  ComponentType;
    NvU16  DeviceID;
    NvU16  DeviceRev;
} GFRMCOMPONENTID, *PGFRMCOMPONENTID;
```

GFRMCOMPONENTID Fields

`ComponentType`	See "ComponentType Definitions" on page 30.
`DeviceID`	Device ID.
`DeviceRev`	Device revision.

The currently defined members of **`ComponentType`** include the following:

ComponentType Definitions

```
#define GF_EPPAPI          0x00000001L
#define GF_GXAPI           0x00000002L
#define GF_VXAPI           0x00000003L
#define GF_JXEAPI          0x00000004L
#define GF_JXDAPI          0x00000005L
#define GF_MXEAPI          0x00000006L
#define GF_MXDAPI          0x00000007L
#define GF_CPUAPI          0x00000008L
#define GF_DXAPI           0x00000009L
#define GF_IXAPI           0x0000000AL
#define GF_ISPAPI          0x0000000BL
#define GF_FDEVAPI         0x0000000CL
#define GF_BDEVSDAPI       0x0000000DL
#define GF_BDEVMDAPI       0x0000000EL
#define GF_UIAPI           0x0000000FL
#define GF_OSXAPI          0x00000010L
#define GF_I2CAPI          0x00000011L
#define GF_I2SAPI          0x00000012L
#define GF_MMPROCAPI       0x00000013L
#define GF_CAMAPI          0x00000014L
#define GF_3DAPI           0x00000015L
#define GF_INTXAPI         0x00000016L
#define GF_MXDH264API      0x00000017L
#define GF_MXEH264API      0x00000018L
#define GF_RMAPI           0x00000019L
#define GF_GXFBAPI         0x0000001AL
#define GF_MXDRV9API       0x0000001BL
#define GF_MXDVC1API       0x0000001CL
```

Surface Manager Services

The surface manager manages a list of surfaces. It works in conjunction with the GFRm memory manager services with respect to embedded memory management.

Surface Overview

A *surface* describes a two-dimensional memory region used by different GFSDK components. It is used heavily by video-related components. The *primary surface* is the main surface that maps to the onscreen frame buffer. It is associated with the main LCD panel display. Alternatively, there could be a *subprimary* surface that is associated with the subLCD panel display. A display always uses the Cartesian coordinate system, and the primary or subprimary surface is the memory region that maps to the two-dimensional region associated with the display.

To access the region associated with a surface, one or more linear pointers with corresponding strides are provided for the user. A surface also contains certain attributes, namely:

- Type of memory (either embedded video memory or system runtime memory)
- Color format for the memory region, whether RGB or different YUV formats.
- Rotation orientation

A primary surface is pre-allocated, meaning it always matches the LCD display panel's dimensions.

Surface Manager Functions

The surface manager functions are as follows:

GFRmSurfaceAlloc()/GFRmSurfaceAllocWithCS()

GFRmSurfaceAlloc() is the surface allocation routine, and **GFRmSurfaceAllocWithCS()** is the equivalent but provides critical section synchronization with hardware access. The functions use the same parameters and return values.

Function Prototype

```
GF_RETTYPE  GFRmSurfaceAlloc(
/* GF_RETTYPE  GFRmSurfaceAllocWithCS( */
    GF_HANDLE           RmHandle,
    GFRMSURFACEREQUEST  *pSurfaceReq,
    PGFRMSURFACE        *ppSurface );
```

Parameters

RmHandle	Handle specific to the GFRm.
*pSurfaceReq	Pointer to surface request structure.
*ppSurface	Pointer to surface pointer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR_NO_SUPPORT	If requested surface format is not supported.
GF_ERROR_OUT_MEMORY	If out of memory.
GF_ERROR_BAD_PARAMETER	If parameter is bad or unrecognizable.
GF_ERROR	If other error.

GFRmSurfaceFree()/GFRmSurfaceFreeWithCS()

GFRmSurfaceFree() frees an allocated surface. **GFRmSurfaceFreeWithCS()** is equivalent to it but provides critical section synchronization with hardware access. The functions have the same parameters and return values.

Function Prototype

```
GF_RETTYPE  GFRmSurfaceFree(
/* GF_RETTYPE  GFRmSurfaceFreeWithCS( */
    GF_HANDLE     RmHandle,
    PGFRMSURFACE  *ppSurface );
```

Parameters

RmHandle	Handle specific to the GFRm.
*ppSurface	Pointer to surface pointer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmSurfaceLock()

Locks a surface. Usually called before accessing the surface.

Function Prototype

```
GF_RETTYPE  GFRmSurfaceLock(
    GF_HANDLE    RmHandle,
    GFRMSURFACE  *pSurface );
```

Parameters

RmHandle	Handle specific to the GFRm.
*pSurface	Pointer to surface structure.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmSurfaceUnlock()

Unlocks a surface and is usually called after accessing the surface.

Function Prototype

```
GF_RETTYPE  GFRmSurfaceUnlock(
    GF_HANDLE    RmHandle,
    GFRMSURFACE  *pSurface );
```

Parameters

RmHandle	Handle specific to the GFRm.
*pSurface	Pointer to surface structure.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmSurfaceQueryPrimaryInfo()

This service provides primary surface information. It returns information that can be used for reference, but the returned surface pointer should not be used for other operations.

Function Prototype

```
GF_RETTYPE  GFRmSurfaceQueryPrimaryInfo(
    GF_HANDLE     RmHandle,
    PGFRMSURFACE  *ppSurface,
    NvU32         *pRotate,
    NvU32         subSurfaceType );
```

Parameters

RmHandle	Handle specific to the GFRm.
*ppSurface	Pointer to the primary surface pointer. Used for quick reference only. *This returned primary surface pointer should not be used for other API operations.*
pRotate	Returns the current primary surface rotation orientation. It can be set to NULL if this information is not required.
subSurfaceType	GF_SURFACE_PRIMARY or GF_SURFACE_PRIMARY_SUB.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmSurfaceUpdate()/GFRmSurfaceUpdateWithCS()

GFRmSurfaceUpdate() updates the surface structure with data pertinent to what is specified in **flag**. **GF_SURFACE_UPDATE_ROTATE**, which causes the surface to be rotated immediately, is the only supported value for **flag**.

GFRmSurfaceUpdateWithCS() is an equivalent function that provides critical section synchronization when accessing hardware.

Function Prototype

```
GF_RETTYPE  GFRmSurfaceUpdate(
/* GF_RETTYPE  GFRmSurfaceUpdateWithCS( */
    GF_HANDLE     RmHandle,
    PGFRMSURFACE  pSurface,
    NvU32         flag,
    NvU32         data );
```

Parameters

RmHandle	Handle specific to the GFRm.
pSurface	Pointer to the surface structure.
flag	GF_SURFACE_UPDATE_ROTATE is supported. See "GFRmSurfaceUpdate() Definitions" on page 35.
data	Data specific to flag.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmSurfaceUpdate() Definitions

```
/*
    Surface Update Flag
*/
#define GF_SURFACE_UPDATE_ROTATE                        1
    // Rotate surface immediately. PassesGF_SURFACE_PRIMARY_0,
    // GF_SURFACE_PRIMARY_90, GF_SURFACE_PRIMARY_180, or
    // GF_SURFACE_PRIMARY_270 as the data parameter in
    // GFRmSurfaceUpdate().
```

Surface Manager Structures

The surface manager structures include these:

GFRMSURFACE

This is the data structure provided to the user by the GFRm that describes a surface. The pointers to the memory region (**pY**, **pU**, and **pV**), together with the corresponding strides, are assigned internally.

GFRMSURFACE Structure

```
typedef struct  _GFRMSURFACE {
    NvU32  width;
    NvU32  height;
    NvU32  SurfaceType;
    NvU32  ColorFormat;
    NvU32  YStride;
    NvU32  UStride;
    NvU32  VStride;
    NvU8   *pY;
    NvU8   *pU;
    NvU8   *pV;
} GFRMSURFACE, *PGFRMSURFACE;
```

GFRMSURFACE Fields

`width`	Surface width in pixels.
`height`	Surface height in lines.
`SurfaceType`	Surface type. See "GFRMSURFACE Surface Type Definitions" on page 37.
`ColorFormat`	Surface color format. See "GFRMSURFACE Color Format Definitions" on page 38.
`YStride;`	Stride for Y plane of YUV4:2:0, YUV4:2:2, or RGB surfaces.
`UStride`	U plane stride (not used in RGB surface).
`VStride`	V plane stride (not used in RGB surface).
`*pY`	Pointer to non-planar surface (e.g., RGB) or Y plane of YUV4:2:0.

GFRMSURFACE Fields (continued)

*pU	Pointer to U plane.
*pV	Pointer to V plane.

GFRMSURFACE Surface Type Definitions

```
#define GF_SURFACE_ROTATE_0                     0x00000000
#define GF_SURFACE_ROTATE_90                    0x10000000
    // Always relative to GF_SURFACE_ROTATE_0.
#define GF_SURFACE_ROTATE_180                   0x20000000
    // Always relative to GF_SURFACE_ROTATE_0.
#define GF_SURFACE_ROTATE_270                   0x30000000
    // Always relative to GF_SURFACE_ROTATE_0.
/*
    GF_SURFACE_H_FLIP and GF_SURFACE_V_FLIP can be ORed with
    the four rotation options above to create 16 combinations.
    In reality, there are only eight unique combinations.
*/
#define GF_SURFACE_H_FLIP                       0x40000000
    // Horizontal flip.
#define GF_SURFACE_V_FLIP                       0x80000000
    // Vertical flip.
#define GF_SURFACE_ROTATE_MASK                  0xF0000000
#define GF_SURFACE_ROTATE_SHIFT                         28
    // Bits[31:28]
#define GF_SURFACE_ROTATE_WRAP                           4
#define GF_SURFACE_PRIMARY                      0x00000001
    // Primary surface.
#define GF_SURFACE_PRIMARY_0                             \
       (GF_SURFACE_PRIMARY| GF_SURFACE_ROTATE_0)
#define GF_SURFACE_PRIMARY_90                            \
       (GF_SURFACE_PRIMARY| GF_SURFACE_ROTATE_90)
#define GF_SURFACE_PRIMARY_180                           \
       (GF_SURFACE_PRIMARY| GF_SURFACE_ROTATE_180)
#define GF_SURFACE_PRIMARY_270                           \
       (GF_SURFACE_PRIMARY| GF_SURFACE_ROTATE_270)
#define GF_SURFACE_PRIMARY_SUB                  0x00000100
    // Subprimary surface.
#define GF_SURFACE_OVERLAY                      0x00000002
#define GF_SURFACE_VIDEO_MEMORY                 0x00000004
#define GF_SURFACE_SYSTEM_MEMORY                0x00000008
```

GFRMSURFACE Surface Type Definitions (continued)

```
#define GF_SURFACE_SHARE                        0x00001000
    // Surface to be allocated from shared embedded memory.
#define GF_SURFACE_PREALLOCATED_MEM             0x00002000
    // Allocate surface with pre-allocated memory pointer.
```

GFRMSURFACE Color Format Definitions

```
/*
    ColorFormat: Describes the format or type of the content
                 pointed to by pointers in surface structure.
*/
#define GF_SURFACE_YUV420                                1
    // Planar
#define GF_SURFACE_YUV422                                2
#define GF_SURFACE_YUV444                                4
#define GF_SURFACE_ROTATED_YUV422                        8
#define GF_SURFACE_YUYV   (GF_SURFACE_YUV422|0x00000004)
#define GF_SURFACE_YVYU   (GF_SURFACE_YUV422|0x00000008)
#define GF_SURFACE_UYVY   (GF_SURFACE_YUV422|0x00000010)
#define GF_SURFACE_VYUY   (GF_SURFACE_YUV422|0x00000020)
#define GF_SURFACE_MPEGDEC(GF_SURFACE_YUV420|0x00000040)
#define GF_SURFACE_MPEGENC                      0x00000080
    // This image is for encoding purposes.
    // Must be combined with a particular YUV format.
#define GF_SURFACE_JPEGDEC (GF_SURFACE_YUV420|0x00000100)
#define GF_SURFACE_JPEGENC                      0x00000200
    // This image is for encoding purposes.
    // Must be combined with a particular YUV format.
#define GF_SURFACE_PLANAR_YUV422                0x00000400
#define GF_SURFACE_RGB565                       0x00010000
#define GF_SURFACE_RGB888                       0x00020000
#define GF_SURFACE_ARGB8888                     0x00030000
    // Not used for GoForce 3000, GoForce 4000.
#define GF_SURFACE_ARGB1555                     0x00040000
    // Only for GoForce 4000 Rev B and later.
#define GF_SURFACE_ARGB4444                     0x00050000
    // Only for GoForce 4000 Rev B and later.
```

GFRMSURFACEREQUEST

This structure is used to request a surface during a surface allocation.

GFRMSURFACEREQUEST Structure

```
typedef struct  _GFRMSURFACEREQUEST {
    NvU32  width;
    NvU32  height;
    NvU32  SurfaceType;
    NvU32  ColorFormat;
    NvU32  hint;
    NvU8   *pY;
    NvU8   *pU;
    NvU8   *pV;
} GFRMSURFACEREQUEST, *PGFSURFACEREQUEST;
```

GFRMSURFACEREQUEST Fields

`width`	Width in pixels.
`height`	Height in lines.
`SurfaceType`	See "GFRMSURFACE Surface Type Definitions" on page 37.
`ColorFormat`	See "GFRMSURFACE Color Format Definitions" on page 38.
`hint`	See "GFRMSURFACEREQUEST Definitions" on page 39.
`*pY`	Pointer to memory that is used to allocate with pre-allocated memory pointer.
`*pU`	Pre-allocated memory pointer.
`*pV`	Pre-allocated memory pointer.

GFRMSURFACEREQUEST Definitions

```
/*
    Surface Hints
*/
#define GF_SURFACE_HINT_TOP_DOWN                    0x1
    // Allocate from highest position (high address).
#define GF_SURFACE_HINT_BOTTOM_UP                   0x2
    // Allocate from lowest possible position (low address).
#define GF_SURFACE_HINT_MAX_CHUNK                   0x4
    // Allocate maximum chunk of memory.
#define GF_SURFACE_HINT_ALIGN32                     0x8
    // Allocate RGB surfaces so stride is 32-bit aligned.
```

Memory Manager Services

These services deal with low-level memory management. They are not usually called by the application layer. For all embedded memory management, the memory manager works closely with the surface manager to ensure proper memory utilization.

Memory Manager Functions and Data Structure

The memory manager has three functions and a data structure:

GFRmMemInfo()

Current information on the available memory is provided by this service. Only embedded memory information is supported.

Function Prototype

```
GF_RETTYPE  GFRmMemInfo(
    GF_HANDLE  RmHandle,
    NvU32      *pTotalFree,
    NvU32      *pMaxChunk,
    NvU32      flag );
```

Parameters

`RmHandle`	Handle specific to the GFRm.
`*pTotalFree`	Pointer to variable containing the total number of free bytes.

Parameters (continued)

*pMaxChunk	Pointer to the variable containing the byte size of the biggest chunk of free memory.
flag	Type of memory. The only supported type is GF_MEMORY_EMBEDDED.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmMemGetOffset()

This function performs a handle-to-offset (SC15 view of hardware address) conversion.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmMemGetOffset(void *MemHandle,
                 NvU32 *offset);
```

Parameters

*MemHandle

*offset

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmMemGetPointer()

Tis function performs a handle-to-virtual address conversion. The returned pointer cannot be dereferenced except in direct addressing mode.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmMemGetPointer(void *MemHandle,
                  void **pointer);
```

Parameters

*MemHandle

**pointer

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmMemOffsetToHandle()

This functions returns the memory handle reverse based on the offset. This can be used only for the embedded memory.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmMemOffsetToHandle(GF_HANDLE RmHandle,
                      NvU32     offset,
                      void    **MemHandle);
```

Parameters

RmHandle

offset

**MemHandle

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmMemOffsetToVirt()

This function converts the offset to a virtual address.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmMemOffsetToVirt(GF_HANDLE RmHandle,
                    NvU32     offset,
                    void    **ppMemory);
```

Parameters

RmHandle	Handle to the resource manager.
offset	Hardware offset as seen by the SC15x chip
**ppMemory	Pointer to the virtual address

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmMemVirtToOffset

This function converts the virtual address to the offset (SC15 hardware view).

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmMemVirtToOffset(GF_HANDLE RmHandle,
                    void     *ppMemory,
                    NvU32    *offset);
```

Parameters

RmHandle	Handle to the resource manager.
**ppMemory	Pointer to the virtual address
*offset	Hardware offset SC15 view

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmMemHandleAlloc()

This function allocates a linear piece of internal memory block, and is maintained for backward compatibility.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmMemAlloc( GF_HANDLE          RmHandle,
              GFRMMEMORYREQUEST *pMemoryReq,
              void              **pMem);
```

Parameters

`RmHandle`

`*pMemoryReg`

`**pMem`

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmMemHandleFree()

This function frees an allocated internal memory block, and is maintained for backward compatibility.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmMemFree( GF_HANDLE  RmHandle,
             void       **pMem);
```

Parameters

`RmHandle`

`**pMem`

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRMMEMORYREQUEST

Data structure used by **`GFRmMemAlloc()`**.

GFRMMEMORYREQUEST Structure

```
typedef struct  _GFRMMEMORYREQUEST {
    NvU32       length;
    NvU32       hint;
    struct
    {
        NvU8  type;
        NvU8  share;
```

GFRMMEMORYREQUEST Structure (continued)

```
        NvU8  align;
        NvU8  reserved;                    // Do not use.
    }
    flag;
} GFRMMEMORYREQUEST, *PGFRMMEMORYREQUEST;
```

GFRMMEMORYREQUEST Fields

`length` Requested length in bytes.

`hint` See "GFRMMEMORYREQUEST Definitions" on page 45.

`flag`

- `type`
  See "GFRMMEMORYREQUEST Definitions".
- `share`
  Shared memory if 1. (*Reserved for future use.*)
- `align`
  See "GFRMMEMORYREQUEST Definitions".

GFRMMEMORYREQUEST Definitions

```
/*
    Memory type
*/
#define GF_MEMORY_SYSTEM                                0x01
#define GF_MEMORY_EMBEDDED                              0x02
#define GF_MEMORY_MEMMAPPED                             0x04
#define GF_MEMORY_SHARE                                 0x80
    // Used in GFRmMemInfo only.
/*
    Memory alignment
*/
#define GF_MEMORY_ALIGN_NONE                             0x0
    // Don't care.
#define GF_MEMORY_ALIGN2                                 0x1
    // Aligned to 2-byte boundary.
#define GF_MEMORY_ALIGN4                                 0x2
    // Aligned to 4-byte boundary.
#define GF_MEMORY_ALIGN8                                 0x3
    // Aligned to 8-byte boundary.
#define GF_MEMORY_ALIGN16                                0x4
    // Aligned to 16-byte boundary.
```

GFRMMEMORYREQUEST Definitions (continued)

```
#define GF_MEMORY_ALIGN32                               0x5
    // Aligned to 32-byte boundary.
#define GF_MEMORY_ALIGN64                               0x6
    // Aligned to 64-byte boundary.
#define GF_MEMORY_ALIGN128                              0x7
    // Aligned to 128-byte boundary.
#define GF_MEMORY_ALIGN256                              0x8
    // Aligned to 256-byte boundary.
/*
    Memory Hint - Follow Surface Hint
*/
#define GF_MEMORY_HINT_TOP_DOWN GF_SURFACE_HINT_TOP_DOWN
    // High address to low address.
#define GF_MEMORY_HINT_BOTTOM_UP GF_SURFACE_HINT_BOTTOM_UP
    // Low address to high address.
#define GF_MEMORY_HINT_MAX_CHUNK GF_SURFACE_HINT_MAX_CHUNK
    // Maximum chunk of memory: Specify -1 for
    // length in GFRMMEMORYREQUEST struct.
#define GF_MEMORY_HINT_FIXED (GF_MEMORY_HINT_MAX_CHUNK << 1)
    // Memory chunk to be fixed permanently.
```

Operating System Manager Services

The operating system manager services are abstractions of services that are dependent on the host operating system.

GFRmOSWaitMSec()

Waits until the number of milliseconds specified by the **msec** parameter elapses. This function is highly operating system dependent and is usually a wrapper for the equivalent function in the operating system.

Function Prototype

```
GF_RETTYPE  GFRmOSWaitMSec(
    GF_HANDLE  RmHandle,
    NvU32      msec );
```

Parameters

RmHandle	Handle specific to the GFRm.
msec	Milliseconds to wait.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmOSGetTickCount()

This service returns the current operating-system-dependent tick counter. To check for tick counter resolution support, call this service with the appropriate resolution parameter first. To simply check for resolution support, set both **hi** and **low** pointers to NULL.

Function Prototype

```
GF_RETTYPE  GFRmOSGetTickCount(
    GF_HANDLE  RmHandle,
    NvU32      *hi,
    NvU32      *low,
    NvU32      resolution );
```

Parameters

RmHandle	Handle specific to the GFRm.
*hi	Pointer to high 32-bit tick count variable.
*low	Pointer to low 32-bit tick count variable.
resolution	Tick counter resolution. See "GFRmOSGetTickCount Definitions" on page 48.

Return Values

GF_SUCCESS	If successful.
GF_ERROR_BAD_PARAMETER	If bad parameter.
GF_ERROR	If not supported or other error.

GFRmOSGetTickCount Definitions

```
#define GF_USEC_TICKS                                   0
    // Microsecond.
#define GF_MSEC_TICKS                                   1
    // Millisecond.
#define GF_SEC_TICKS                                    2
    // Second.
```

GFRmOSEnterCS()

This function is called to enter a critical section. This function is used in a multithread environment to synchronize GoForce hardware access. This function is highly operating system dependent and is usually a wrapper for the equivalent function in the operating system

Function Prototype

```
GF_RETTYPE  GFRmOSEnterCS(
    GF_HANDLE  RmHandle );
```

Parameters

RmHandle	Handle specific to the GFRm.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmOSEnterCSExt()

This function is a superset of **GFRmOSEnterCS()**. It takes an additional critical section **id** parameter. It can be used for entering different critical sections used in the GFSDK software for multiprocessing environments.

Function Prototype

```
GF_RETTYPE  GFRmOSEnterCS(
```

Function Prototype (continued)

```
    GF_HANDLE  RmHandle;
    NvU32      id );
```

Parameters

RmHandle	Handle specific to the GFRm.
id	Critical section ID. See "GFRmOSEnterCSExt()/ GFRmOSExitCSExt() Definitions".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmOSEnterCSExt()/GFRmOSExitCSExt() Definitions

```
/*
    Operating System Critical Section: Advanced Usage Only.
*/
#define GF_CRITICAL_SECTION_HW                0x00000001
#define GF_CRITICAL_SECTION_SW1               0x00000002
#define GF_CRITICAL_SECTION_SW2               0x00000004
#define GF_CRITICAL_SECTION_SW3               0x00000008
#define GF_CRITICAL_SECTION_SW4               0x00000010
#define GF_CRITICAL_SECTION_SW5               0x00000020
#define GF_CRITICAL_SECTION_SW6               0x00000040
#define GF_CRITICAL_SECTION_SW7               0x00000080
#define GF_CRITICAL_SECTION_SW8               0x00000100
#define GF_CRITICAL_SECTION_DSP               0x00000200
```

GFRmOSExitCS()

GFRmOSExitCS() is called to exit a critical section and is used in a multithread environment to synchronize GoForce hardware access. This function is highly dependent on the operating system and is usually a wrapper for the equivalent function in the operating system.

Function Prototype

```
GF_RETTYPE  GFRmOSExitCS(
    GF_HANDLE  RmHandle );
```

Parameters

RmHandle	Handle specific to the GFRm.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmOSExitCSExt()

This function is an extended superset of **GFRmOSExitCS()**. It takes an additional critical section **id** parameter. It can be used to exit other critical sections used in the GFSDK software for multiprocessing environments.

Function Prototype

```
GF_RETTYPE  GFRmOSExitCSExt(
    GF_HANDLE  RmHandle;
    NvU32      id );
```

Parameters

RmHandle	Handle specific to the GFRm.
id	Critical section ID. See "GFRmOSEnterCSExt()/ GFRmOSExitCSExt() Definitions" on page 49.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmOSCS()

This is the general critical section function that exposes critical section services specific to an operating system. The operations involved include create, destroy, lock, and unlock. The parameter **id** can be chosen by the system builder to be specific to a given platform.

Function Prototype

```
GF_RETTYPE  GFRmOSCS(
    GF_HANDLE  RmHandle,
    NvU32      id,
```

Function Prototype (continued)

```
    NvU32      operation,
    GF_HANDLE  *pCS );
```

Parameters

RmHandle	Handle specific to the GFRm.
id	Either pre-defined by the resource manager, as with GFRmOSEnterCSExt(), or operating system specific and chosen by the system builder (and may return an error if not supported).
operation	See "GFRmOSCS() Definitions".
pCS	Pointer to critical section object.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.
GF_ERROR_BAD_PARAMETER	If wrong parameter.

GFRmOSCS() Definitions

```
#define  GF_CRITICAL_SECTION_CREATE          0x00000001
#define  GF_CRITICAL_SECTION_DESTROY         0x00000002
#define  GF_CRITICAL_SECTION_LOCK            0x00000004
#define  GF_CRITICAL_SECTION_UNLOCK          0x00000008
```

Interface Manager Services

These services provide accesses to internal register I/O and embedded memory space. All direct and indirect addressing is to be covered here.

Interface Manager Functions and Address Flags

The address flags (see "Interface Manager Address Flags" on page 57) are used by the three functions of the interface manager.

- "GFRmInterfaceGetAddress()" on page 52
- "GFRmInterfaceGetWrite()" on page 53
- "GFRmInterfaceGetRead()" on page 55

GFRmInterfaceGetAddress()

The parameter **`*pFlag`** is used to select the type of address space for the NVIDIA hardware device. It can be used to decide which device's address space is to be used too. Except for the **`GF_ADDR_INDIRECT`** case, all types are likely to be memory mapped if they are running on a virtual memory system. If the resource manager detects that the hardware only supports indirect addressing for that address space, it sets the **`GF_ADDR_INDIRECT`** bit in the returned **`*pFlag`** variable.

Function Prototype

```
GF_RETTYPE  GFRmInterfaceGetAddress(
    GF_HANDLE  RmHandle,
    NvU32      *pFlag,
    NvU16      DeviceID,
    NvU16      DeviceRev,
    void       **ppAddress );
```

Parameters

`RmHandle`	Handle specific to the GFRm.
`*pFlag`	Address flag.
`DeviceID`	Targeted device ID.
`DeviceRev`	Targeted device revision.
`**ppAddress`	Pointer to address.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmInterfaceGetWrite()

The parameter **flag** is used to select the address space for the NVIDIA hardware device. It can be used to decide which device's address space is to be used too. The returned **ppWrite** function pointer takes data and an offset for **GFRmInterfaceGetWrite()**.

Function Prototype

```
GF_RETTYPE  GFRmInterfaceGetWrite(
    GF_HANDLE  RmHandle,
    NvU32      flag,
    NvU16      DeviceID,
    NvU16      DeviceRev,
    void       **ppWrite );
```

Parameters

RmHandle	Handle specific to the GFRm.
flag	Address flag.
DeviceID	Targeted device ID.
DeviceRev	Targeted device revision.
**ppWrite	Pointer to a write function. See "GFRmInterfaceGetWrite()/ GetWriteEx() Definitions" on page 54.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

The parameter for the `typedef`s below, `ptr_handle`, is the base offset pointer for I/O or embedded memory if the function pointer is returned by **`GFRmInterfaceGetWrite()`**.

GFRmInterfaceGetWrite()/GetWriteEx() Definitions

```
/*
    Interface FIFO Flags
*/
#define GF_FIFO_FEND                            0x00000001
    // Front-End FIFO.
#define GF_FIFO_SRC                             0x00000002
    // GE (2D/Primary) FIFO.
/*
    Interface Get-Write Function
*/
typedef void (*GF_FUNC_WRITE_32BIT)
             (void *ptr_handle, void* ptr, NvU32 data);
typedef void (*GF_FUNC_WRITE_16BIT)
             (void *ptr_handle, void* ptr, NvU16 data);
typedef void (*GF_FUNC_WRITE_8BIT)
             (void *ptr_handle, void* ptr, NvU8 data);
/*
    Interface Get-Read-(Modified)-Write Function
*/
typedef void (*GF_FUNC_READWRITE_32BIT)
             (void *ptr_handle, void* ptr, NvU32 orData,
              NvU32 andMask);
typedef void (*GF_FUNC_READWRITE_16BIT)
             (void *ptr_handle, void* ptr, NvU16 orData,
              NvU16 andMask);
typedef void (*GF_FUNC_READWRITE_8BIT)
             (void *ptr_handle, void* ptr, NvU8 orData,
              NvU8 andMask);
/*
    Interface Get-FillEmbedded Function
*/
typedef void (*GF_FUNC_FILL_EMBEDDED_32BIT)
             (void *ptr_handle, void* ptr, NvU32 sizeIn32bit,
              NvU32 data);
typedef void (*GF_FUNC_FILL_EMBEDDED_16BIT)
             (void *ptr_handle, void* ptr, NvU32 sizeIn16bit,
              NvU16 data);
```

GFRmInterfaceGetWrite()/GetWriteEx() Definitions (continued)

```
typedef void (*GF_FUNC_FILL_EMBEDDED_8BIT)
             (void *ptr_handle, void* ptr, NvU32 sizeIn8bit,
              NvU8 data);
/*
    Interface Get-WriteEmbedded Function
*/
typedef void (*GF_FUNC_WRITE_EMBEDDED)
             (void *ptr_handle, void* srcPtr, void* dstPtr,
              NvU32 sizeInBytes);
/*
    Interface Get-WriteFIFO Function
*/
typedef void (*GF_FUNC_WRITE_FIFO)
             (void *ptr_handle, void* srcPtr,
              NvU32 srcSizeIn32bits, void * dstPtr,
              NvU32 dstSizeIn32bits);
typedef void (*GF_FUNC_WRITE_EMBEDDED_FIFO)
             (void *ptr_handle, void* srcPtr,
              NvU32 srcSizeIn32bits, void* dstPtr,
              NvU32 dstSizeIn32bits, NvU32 option);
```

GFRmInterfaceGetRead()

The parameter **flag** is used to select the address space for the NVIDIA hardware device. It can be used to decide which device's address space is to be used too. The returned **ppRead** function pointer takes an offset for **GFRmInterfaceGetRead()** and returns the read data.

Function Prototype

```
GF_RETTYPE  GFRmInterfaceGetRead(
    GF_HANDLE  RmHandle,
    NvU32      flag,
    NvU16      DeviceID,
    NvU16      DeviceRev,
    void       **ppRead );
```

Parameters

RmHandle	Handle specific to the GFRm.
flag	Address flag.
DeviceID	Targeted device ID.

Parameters (continued)

DeviceRev	Targeted device revision.
**ppRead	Pointer to a read function. See "GFRmInterfaceGetRead()/ GetReadEx() Definitions" on page 56.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

The parameter for the typedefs below, ptr_handle, is the base offset pointer for I/O or embedded memory if the function pointer is returned by **GFRmInterfaceGetRead()**.

GFRmInterfaceGetRead()/GetReadEx() Definitions

```
/*
    Interface Get-Read Function
*/
typedef NvU32 (*GF_FUNC_READ_32BIT)
              (void *ptr_handle, void *ptr);
typedef NvU16 (*GF_FUNC_READ_16BIT)
              (void *ptr_handle, void *ptr);
typedef NvU8 (*GF_FUNC_READ_8BIT)
             (void *ptr_handle, void *ptr);
/*
    Interface Get-ReadEmbedded Function
*/
typedef void (*GF_FUNC_READ_EMBEDDED)
             (void *ptr_handle, void *srcPtr, void *dstPtr,
              NvU32 sizeInBytes);
/*
    Interface Get-ReadFIFO Function
*/
typedef void (*GF_FUNC_READ_FIFO)
             (void *ptr_handle, void *srcPtr,
              NvU32 srcSizeIn32bits, void *dstPtr,
              NvU32 dstSizeIn32bits);
```

Interface Manager Address Flags

These flags are used by the interface manager functions.

Interface Manager Address Flag Definitions

```
#define GF_ADDR_START                           0x00000001
#define GF_ADDR_IO                              0x00000002
#define GF_ADDR_EMBEDDED_MEM                    0x00000004
#define GF_ADDR_FRONTEND_FIFO                   0x00000008
    // I/O and memory write via front-end FIFO.
#define GF_ADDR_READ_WRITE                      0x00000010
    // Read-modify-write.
#define GF_ADDR_FILL                            0x00000020
    // Must set ADDR_EMBEDDED_MEM.
#define GF_ADDR_COPY                            0x00000040
    // Must set ADDR_EMBEDDED_MEM.
#define GF_ADDR_FIFO                            0x00000080
    // Must set ADDR_IO.
#define GF_ADDR_32BIT                           0x00000100
    // Must select ADDR_IO or _MEM.
#define GF_ADDR_16BIT                           0x00000200
    // Must select ADDR_IO or _MEM.
#define GF_ADDR_8BIT                            0x00000400
    // Must select ADDR_IO or _MEM.
#define GF_ADDR_DIRECT                          0x00001000
#define GF_ADDR_INDIRECT8                       0x00002000
#define GF_ADDR_INDIRECT16                      0x00004000
#define GF_ADDR_DEVICE                          0x00008000
#define GF_ADDR_DEVICE_ID_MASK                  0xFFFF0000
#define GF_ADDR_BIT_MASK                                \
       (GF_ADDR_32BIT|GF_ADDR_16BIT|GF_ADDR_8BIT)
#define GF_ADDR_MASK                                    \
       (GF_ADDR_DIRECT|GF_ADDR_INDIRECT8|GF_ADDR_INDIRECT16)
```

Utility Manager Services

GFRmUtilWaitKey() is the only service available.

GFRmUtilWaitKey()

When a key press is detected, this function returns the virtual key code. If it times out, it returns 0.

Function Prototype

```
int  GFRmUtilWaitKey(
    GF_HANDLE  RmHandle,
    NvU32      timeOutMSec );
```

Parameters

RmHandle	Handle specific to the GFRm.
timeOutMSec	Milliseconds until time out. If 0, wait until a key is pressed.

Return Values

Virtual key code	If key press is detected.
0	If no key is pressed and it times out.
GF_ERROR	If error.

Helper Services

Helper services are designed to isolate those library functions that are specific to a given programming language in order to enhance code portability. Helper services do not need to specify a GFRm handle. These services are mostly C language macros that wrap library functions found within the operating system and compiler suites.

All helper services start with the prefix **GF** instead of **GFRm**.

- String helpers.

  These are macros that usually follow standard C functions. There are also wide character versions (usually for UNICODE strings), such as **GFWStrcpy()** for **GFStrcpy()**. These helper services are as follows:

  - **GFStrcpy()**, **GFWStrcpy()**

- **GFStrncpy(),GFWStrncpy()**
- **GFStrncat(),GFWStrncat()**
- **GFStrlen(),GFWStrlen()**
- **GFStrchr(),GFWStrchr()**
- **GFStrcmp(),GFWStrcmp()**
- **GFStricmp(),GFWStricmp()**
- **GFStrincmp() // GFStrncmp that is case insensitive**
- **GFStrncmp(), GFWStrncmp()**
- **GFToupper()**
- **GFStrConvert(), GFWStrConvert()**

- System memory helpers.

  These are macros that follow the standard C style.

  - **GFMalloc()**
  - **GFFree()**
  - **GFMemcpy()**
  - **GFMemcmp()**
  - **GFMemset()**

- Miscellaneous helpers.

  These are debugging service and data types.

  - **GFPrintf()**
  - **GFWCHAR()**
  - **GFWTEXT()**

GFRm Programming

This section discusses options for consistent GFSDK coding and the general programming sequence for the GFRm.

Options

The top-level `GFOption.h` file contains information specific to platforms and compilers that can be used by different GFSDK components and applications for consistency. If the options in this file are adhered to, code for the GFSDK can be kept portable across different platforms.

General Programming Sequences

For starters, the best way to use and to learn the GFRm is to refer to the sample application source code. A new application can be started by creating a project at the same directory level as the sample application source directory. This way the same directory tree structure can be maintained. The general sequence for programming the GFRm is as follows:

1. Make sure that the build project or file has the right paths to the library and header files. Use relative directory addressing if possible. The **`GFSDK\inc`** directory must be included.
2. Include `GF.h` in the source file. This header is all that is needed to access GFRm services. This header file includes `GFRm.h`, which has all the structures and function prototypes for accessing GFRm services.
3. Call **`GFRmOpen()`** before any other GFRm functions. Use the **`GF_HANDLE`** returned from this call for all subsequent GFRm functions.
4. Call **`GFRmGetProperty()`** to determine the version being used and to check on other properties.
5. Make calls to the appropriate GFRm functions. Always remember to pass the same **`GF_HANDLE`** returned from **`GFRmOpen()`**.
6. When the application is exited, **`GFRmClose()`** must be called.

CHAPTER

3

Resource Manager Ix (GFRmIx)

Resource Manager Services

Resource Manager Ix services are only available for SC15. It is a set of low-level functions that deal with chip initialization, clocks, and power plane. For SC15, the services are mapped to corresponding GFIxAPI functions. The resource manager Ix services (GFRmIx) comprise the following groups of services:

General GFRmIx Functions

There is one general GFRmIx function:

- "GFRmIxGetProperty() and GFRmIxGetPropertyWithCS()"

GFRmIxGetProperty() and GFRmIxGetPropertyWithCS()

This function returns the properties for this instance of the resource manager. Check the capability flags to determine what is available.

Function Prototype

```
GF_RETTYPE  GFRmIxGetProperty(
/* GF_RETTYPE GFRmIxGetPropertyWithCS( */
    GF_HANDLE    RmHandle,
    PGFPROPERTY  pProp );
```

Parameters

RmHandle	Handle specific to the GFRmIx.
pProp	Pointer to "GFPROPERTY Structure" on page 10.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Host Interface and Clock-Related Functions

The following are the host interface and clock-related functions:

- "GFRmIxInit() and GFRmIxInitWithCS()"
- "GFRmIxDeInit() and GFRmIxDeInitWithCS()"
- "GFRmIxEnableModuleClock() and GFRmIxEnableModuleClockWithCS()"
- "GFRmIxEnableModule() and GFRmIxEnableModuleWithCS()"
- "GFRmIxSetFrequency() and GFRmIxSetFrequencyWithCS()"
- "GFRmIxGetFrequency() and GFRmIxGetFrequencyWithCS()"
- "GFRmIxSetModuleFrequency() and GFRmIxSetModuleFrequencyWithCS()"
- "GFRmIxGetModuleFrequency() and GFRmIxGetModuleFrequencyWithCS()"
- "GFRmIxPowerPlane() and GFRmIxPowerPlaneWithCS()"
- "GFRmIxPowerPlaneHW() and GFRmIxPowerPlaneHWWithCS()"
- "GFRmIxEnableClockSource() and GFRmIxEnableClockSourceWithCS()"
- "GFRmIxSetPLLReferenceClocks() and GFRmIxSetPLLReferenceClocksWithCS()"
- "GFRmIxSetModuleConfig() and GFRmIxSetModuleConfigWithCS()"
- "GFRmIxGetModuleConfig() and GFRmIxGetModuleConfigWithCS()"
- "GFRmIxEnableClock() and GFRmIxEnableClockWithCS()"
- "GFRmIxGetModuleState() and GFRmIxGetModuleStateWithCS()"
- "GFRmIxSelectModuleClock() and GFRmIxSelectModuleClockWithCS()"
- "GFRmIxGPIO() and GFRmIxGPIOWithCS()"

GFRmIxInit() and GFRmIxInitWithCS()

GFRmIxInit() is the first function that accesses the hardware. It initializes certain modules' clocks, takes the modules out of reset, and also initializes device-control registers.

Function Prototype

```
GF_RETTYPE GFRmIxInit(
/* GF_RETTYPE GFRmIxInitWithCS( */
    GF_HANDLE RmHandle )
```

Parameters

RmHandle	Handle specific to the GFRm

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxDeInit() and GFRmIxDeInitWithCS()

GFRmIxDeInit() is the inverse of GFRmIxInit(), and shut downs the chip. All modules are reset and clocks are turned off.

Function Prototype

```
GF_RETTYPE GFRmIxDeInit(
/* GF_RETTYPE GFRmIxDeInitWithCS( */
GF_HANDLE RmHandle )
```

Parameters

RmHandle	Handle specific to the GFRm

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxEnableModuleClock() and GFRmIxEnableModuleClockWithCS()

This function enables and disables a module's clock, and enables the module's source clock. When performing the enable operation and the clock is currently disabled, GFRmIxEnableModuleClock attempts to find the best clock source and divider for the module.

There are two types of enables—the normal enable and the force enable. When doing a normal enable, the module's clock-enable reference count is increased and a call is made to GFRmIxEnableClockSource() which ensures the module's source clock is enabled. When doing a force enable/disable, the modules reference counter is not touched, and a call is made to GFRmIxEnableClockSource() which increments or decrements the source's reference count.

Function Prototype

```
GF_RETTYPE GFRmIxEnableModuleClock(
/* GF_RETTYPE GFRmIxEnableModuleClockWithCS( */
     GF_HANDLE    RmHandle,
     eGFModuleIDs modid,
     NvU32        option,
     NvU32        special )
```

Parameters

`RmHandle`	Handle specific to the GFRm
`eGFModuleIDs`	ID of the module
`option`	Bit field of options, GFIX_ENABLECLOCK, GFIX_DISABLECLOCK, GFIX_FORCEENABLECLOCK, GFIX_FORCEDISABLECLOCK, GFIX_DYNAMICSWITCH, GFIX_SPECIAL
`special`	Specifies a module specific setting, only used when option contains GFIX_SPECIAL

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmIxEnableModule() and GFRmIxEnableModule-WithCS()

This function takes a module out of reset. Each module has a reference count that tracks how many outstanding enables the module has. The module will not be disabled until the reference counter is returned to zero.

Function Prototype

```
GF_RETTYPE GFRmIxEnableModule(
/* GF_RETTYPE GFRmIxEnableModuleWithCS( */
     GF_HANDLE    RmHandle,
     eGFModuleIDs modid,
     NvU32        option,
     NvU32        ComponentType,
     GF_HANDLE    ComponentHandle )
```

Parameters

`RmHandle`	Handle specific to the GFRm
`modid`	ID of the module
`option`	If option is nonzero the module will be enabled
`ComponentType`	Type of component requesting this operation. Extra information included only to be passed to platform code
`ComponentHandle`	Handle to component requesting this operation. Extra information included only to be passed to platform code

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmIxSetFrequency() and GFRmIxSetFrequency-WithCS()

This function sets a clock to a certain frequency. If the clock cannot meet the requested frequency, then the closest one will be chosen. For PLLs, this function determines the closest dividers to arrive at the requested frequency. To determine the frequency that will be chosen without actually setting it, specify GFIX_PLL_QUERY along with the clock.

Some clocks have only one frequency that never changes. In this case, the function must still be called at least once to allow the internal logic to know what the frequency of the clock is. To disable the selection of a clock by a module, a clock can be disabled by setting its frequency to zero; this will remove it from consideration for all modules.

This function will fail if a module is already using the clock.

Function Prototype

```
GF_RETTYPE GFRmIxSetFrequency(
/* GF_RETTYPE GFRmIxSetFrequencyWithCS( */
    GF_HANDLE RmHandle,
    NvU32 option,
    NvU32 *pFrequency )
```

Parameters

`RmHandle`	Handle specific to the GFRm
`option`	GFIX_PLL1, GFIX_PLL2, GFIX_ROSC, GFIX_OSC, GFIX_REFCLK0, GFIX_REFCLK1 or GFIX_PLL_QUERY
`*pFrequency`	Pointer to 32-bit value in KHz to get three digit MHz precision. Returns with newly set current frequency.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmIxGetFrequency() and GFRmIxGetFrequency-WithCS()

This function retrieves the frequency of a clock .

Function Prototype

```
GF_RETTYPE GFRmIxGetFrequency(
/* GF_RETTYPE GFRmIxGetFrequencyWithCS( */
    GF_HANDLE RmHandle,
    NvU32 option,
    NvU32 * pFrequency )
```

Parameters

`RmHandle`	Handle specific to the GFRm
`option`	GFIX_PLL1, GFIX_PLL2, GFIX_ROSC, GFIX_OSC, GFIX_REFCLK0, GFIX_REFCLK1
`*pFrequency`	Pointer to 32-bit value in KHz to get three digit MHz precision. Returns with currently set frequency.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmIxSetModuleFrequency() and GFRmIxSetModule-FrequencyWithCS()

This function sets the desired frequency of a module. The frequency must lie between the minimum and maximum frequencies specified in GFRmIxSetModuleConfig(). This value is cached and used when GFRmIxEnableModuleClock() is called attempting to enable the clock for the module. .

Function Prototype

```
GF_RETTYPE GFRmIxSetModuleFrequency(
/* GF_RETTYPE GFRmIxSetModuleFrequencyWithCS( */
    GF_HANDLE     RmHandle,
```

Function Prototype

```
    eGFModuleIDs  modid,
    NvU32         frequency )
```

Parameters

RmHandle	Handle specific to the GFRm
modid	ID of the module
Frequency	32-bit value in KHz to get three digit MHz precision

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxGetModuleFrequency() and GFRmIxGetModule-FrequencyWithCS()

This function retrieves clock frequency information about a module.

Function Prototype

```
GF_RETTYPE GFRmIxGetModuleFrequency(
/* GF_RETTYPE GFRmIxGetModuleFrequencyWithCS( */
    GF_HANDLE             RmHandle,
    eGFModuleIDs          modid,
    GFIX_MODULEFREQ_TYPE  type,
    NvU32                *pFrequency )
```

Parameters

RmHandle	Handle specific to the GFRm
modid	ID of the module
type	Type of frequency
*pFrequency	Pointer to 32-bit value in KHz to get three-digit MHz precision. Returns with the current frequency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxPowerPlane() and GFRmIxPowerPlaneWithCS()

This function enables and disables power plane(s). Multiple power planes can be disabled in one function call. There are two versions of this function as specified by the compilation flag GF_EXTERNAL_POWERPLANE_LOGIC. If this flag is set to 1 then all the power plane logic is performed in the platform code. If set to 0 then all the logic is performed in Ix.

When internal power plane logic is chosen, each power plane has a reference count. If the power plane is to be enabled or disabled, GFRmIxPowerPlane() will call RmPowerPlane() to perform platform-specific enabling.

When external power plane logic is chosen, all calls to GFRmIxPowerPlane() are forwarded to the platform code, and the GFSDK does not perform synchronization. The platform code is then responsible for reference counting and programming the registers. The function GFRmIxPowerPlaneHW() may be used to do this programming.

Function Prototype

```
GF_RETTYPE GFRmIxPowerPlane(
/* GF_RETTYPE GFRmIxPowerPlaneWithCS( */
    GF_HANDLE                RmHandle,
    NvU32                    ppID,
    GFIX_POWERPLANE_OP_TYPE  operation,
    NvU32                   *states,
    NvU32                    ComponentType,
    GF_HANDLE                ComponentHandle,
    eGFModuleIDs             modid )
```

Parameters

RmHandle	Handle specific to the GFRm
ppID	Bit field of power planes to be operated on

`operation`	Operation to perform on the power planes
`*states`	Ptr to 32 bit value to store the resulting bit field of a GFIX_POWERPLANE_OP_QUERY operation.
`ComponentType`	Type of component requesting this operation. Extra information included only to be passed to platform code
`ComponentHandle`	Handle to component requesting this operation. Extra information included only to be passed to platform code
`modid`	ID of the module requesting this operation. Extra information included only to be passed to platform code

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.
`GF_ERROR_BAD_PARAMETER`	If a bad parameter was passed..

GFRmIxPowerPlaneHW() and GFRmIxPowerPlaneHW-WithCS()

Used internally. This function performs the hardware accesses associated with a GFRmIxPowerPlane() call.

Function Prototype

```
GF_RETTYPE GFRmIxPowerPlaneHW(
/* GF_RETTYPE GFRmIxPowerPlaneHWWithCS( */
    GF_HANDLE RmHandle,
    NvU32     ppID,
    NvU32     option )
```

Parameters

`RmHandle`	Handle specific to the GFRm
`ppID`	ID of power plane
`option`	0 for disable
	1 for enable

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmIxEnableClockSource() and GFRmIxEnableClock-SourceWithCS()

This function enables and disables a clock's source. A reference counter is used to ensure the clock's source won't be disabled until there is a matching number of enable and disable operations. If a clock's source is to be enabled or disabled, GFRmIxEnableClockSource() will call RmEnableClockSource() to do the platform-specific enabling or disabling.

Function Prototype

```
GF_RETTYPE GFRmIxEnableClockSource(
/* GF_RETTYPE GFRmIxEnableClockSourceWithCS(  */
    GF_HANDLE RmHandle,
```

Function Prototype

```
    NvU32     clockid,
    NvU32     state )
```

Parameters

RmHandle	Handle specific to the GFRm
clockid	ID of the clock
state	0 for disable
	1 for enable

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxSetPLLReferenceClocks() and GFRmIxSetPLL-ReferenceClocksWithCS()

This function sets the list of reference clocks that a PLL may choose.

Function Prototype

```
GF_RETTYPE GFRmIxSetPLLReferenceClocks(
/* GF_RETTYPE GFRmIxSetPLLReferenceClocksWithCS( */
    GF_HANDLE RmHandle,
    NvU32     option,
    NvU32     clocks )
```

Parameters

RmHandle	Handle specific to the GFRm
option	GFIX_PLL1, GFIX_PLL2
clocks	Bitfield of clocks. (1<<GFIX_OSC), (1<<GFIX_ROSC),

(1<<GFIX_REFCLK0), (1<<GFIX_REFCLK1)

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxSetModuleConfig() and GFRmIxSetModuleConfigWithCS()

This function sets the bounds and clocking profile for a module. While this function may be called while the module's clock is enabled, it will not affect the current clock settings of the module, All new Ix calls dealing with the module will follow the configuration set by this function.

Function Prototype

```
GF_RETTYPE GFRmIxSetModuleConfig(
/* GF_RETTYPE GFRmIxSetModuleConfigWithCS(  */
    GF_HANDLE           RmHandle,
    eGFModuleIDs        modid,
    GFIXMODULECONFIG    config )
```

Parameters

RmHandle	Handle specific to the GFRm
modid	ID of the module
GFIXMODULECONFIG	Module configuration

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxGetModuleConfig() and GFRmIxGetModuleConfigWithCS()

This function retrieves the bounds and clocking profile for a module.

Function Prototype

```
GF_RETTYPE GFRmIxGetModuleConfig(
/* GF_RETTYPE GFRmIxGetModuleConfigWithCS(  */
    GF_HANDLE          RmHandle,
    eGFModuleIDs       modid,
    GFIXMODULECONFIG   *pconfig )
```

Parameters

RmHandle	Handle specific to the GFRm
modid	ID of the module
*pconfig	Pointer to the module configuration

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxEnableClock() and GFRmIxEnableClockWithCS()

This function performs a master clock enable and disable. When doing a master disable, all the module's clocks currently enabled are disabled. To restore the system to its previous state a master enable should be performed.

Function Prototype

```
GF_RETTYPE GFRmIxEnableClock(
/* GF_RETTYPE GFRmIxEnableClockWithCS( */
    GF_HANDLE RmHandle,
    NvU32     option )
```

Parameters

RmHandle	Handle specific to the GFRm

option	0 for master disable 1 for master enable

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxGetModuleState() and GFRmIxGetModuleState-WithCS()

This function queries information about a module's state.

Function Prototype

```
GF_RETTYPE GFRmIxGetModuleState(
/* GF_RETTYPE GFRmIxGetModuleStateWithCS( */
    GF_HANDLE     RmHandle,
    eGFModuleIDs  modid,
    NvU32        *pState )
```

Parameters

RmHandle	Handle specific to the GFRm
modid	ID of the module
*pState	Pointer to the GFIXMODULESTATE structure in which to store the information

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIxSelectModuleClock() and GFRmIxSelectModuleClockWithCS()

This function forces the selection of a certain clock by a module. The clock most be a valid clock for the module—the list of valid clocks for a module are set using GFRmIxSetModuleConfig(). This function most be called before the module's clock is enabled or the function call will fail.

Function Prototype

```
GF_RETTYPE GFRmIxSelectModuleClock(
/* GF_RETTYPE GFRmIxSelectModuleClockWithCS( */
    GF_HANDLE    RmHandle,
    eGFModuleIDs modid,
    NvU32        clockid )
```

Parameters

`RmHandle`	Handle specific to the GFRm
`modid`	ID of the module
`clockid`	ID of the clock

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmIxGPIO() and GFRmIxGPIOWithCS()

This function manipulates the GPIO pin signals.

Function Prototype

```
GF_RETTYPE GFRmIxGPIO(
/* GF_RETTYPE GFRmIxGPIOWithCS( */
    GF_HANDLE      RmHandle,
    GFIX_GPIO_TYPE gpio,
    NvU32          operation )
    PGFGPIOSTATUS  pGPIOStatus
```

Parameters

`RmHandle`	Handle specific to the GFRm
`gpio`	One of the "GFIX_GPIO_TYPE" enumerated values
`operation`	Combination of the operations described in "Host GPIO Operations" and "GPIO Output Configuration Definitions".
pGPIOStatus	Pointer to the return status information. Refer to "GFGPIOSTATUS" on page 16.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

Attribute Functions

The attribute-related GFRmIx functions are the following:

- "GFRmIxSetAttribute() and GFRmIxSetAttributeWithCS()"
- "GFRmIxGetAttribute() and GFRmIxGetAttributeWithCS()"

GFRmIxSetAttribute() and GFRmIxSetAttributeWithCS()

Function Prototype

```
GF_RETTYPE GFRmIxSetAttribute(
/* GF_RETTYPE GFRmIxSetAttributeWithCS( */
   GF_HANDLE      RmHandle,
   GFIX_ATTR_TYPE attrType,
   NvU32          attrData );
```

Parameters

`RmHandle`	Handle specific to the GFRm
`attrType`	Attribute type
`attrData`	Attribute data

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmIxGetAttribute() and GFRmIxGetAttribute-WithCS()

Function Prototype

```
GF_RETTYPE GFRmIxGetAttribute(
/* GF_RETTYPE GFRmIxGetAttributeWithCS( */
   GF_HANDLE       RmHandle,
   GFIX_ATTR_TYPE  attrType,
   NvU32          *pAttrData );
```

Parameters

RmHandle	Handle specific to the GFRm
attrType	Attribute type
*pAttrData	Pointer to attribute data

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmIx Data Types

Structures and Enumerations

GFIX_MODULEFREQ_TYPE

GFIX_MODULEFREQ_TYPE is an enumerated type that specifies a type of module frequency.

GFIX_MODULEFREG_TYPE enum

```
typedef enum _GFIX_MODULEFREQ_TYPE
 {
    GFIX_MODULEFREQ_OPTIMAL = 0,
       // Used as desired frequency if the desired frequency is
       // not set
    GFIX_MODULEFREQ_MINIMUM,
       // Lowest allowed frequency
    GFIX_MODULEFREQ_MAXIMUM,
       // Highest allowed frequency
    GFIX_MODULEFREQ_REQUESTED,
       // The desired frequency of the module
    GFIX_MODULEFREQ_ACTUAL
       // The actual running frequency of the module
} GFIX_MODULEFREQ_TYPE;
```

GFIXMODULECONFIG

GFIXMODULECONFIG is a structure that is used to specify clocking characteristics of a module.

GFIXMODULECONFIG structure

```
typedef struct _GFIXMODULECONFIG
{
    NvU8    clockSelection;    /**< Selection mode */
    NvU32   minFrequency;      /**< Lowest frequency allowed */
    NvU32   optimalFrequency;  /**< Optimal frequency */
    NvU32   maxFrequency;      /**< Max frequency */
```

GFIXMODULECONFIG structure

```
    NvU32   sourceClocks;      /**< Bitfield of valid source
                                   clocks */
} GFIXMODULECONFIG, *PGFIXMODULECONFIG;
```

GFIXMODULESTATE

GFIXMODULESTATE is a structure that specifies various states of the module.

GFIXMODULESTATE

```
 typedef struct _GFIXMODULESTATE
{
    NvU32   clkRefCount;     /**< Reference counter used in
                                   GFRmIxEnableModuleClock */
    NvU32   clkSelect;       /**< Clock selected for module
                                  (only valid when clkRefCount
                                   is non-zero) */
                             /**< If clkSelect==GFIX_NO_CLOCK,
                                   no clock is selected by
                                   this module */
    NvU32   enableRefCount;  /**< Reference counter used in
                                   GFRmIxEnableModule */
} GFIXMODULESTATE, *PGFIXMODULESTATE;
```

GF_ATTR_TYPE

GF_ATTR_TYPE is an enumerated type that defines the attribute IDs for Set/GetAttributes().

```
typedef enum _GFIX_ATTR_TYPE
 {
    GFIX_ATTR_NONE = 0,
    GFIX_ATTR_DEVICE_INFO,
    GFIX_ATTR_DEVICE_INFO_STRUCT,   /* Get Device ID & Rev */
    GFIX_ATTR_EFUSE_PRODUCT_SKU_ID
} GFIX_ATTR_TYPE;
```

GFIX_MODULEFREQ_TYPE

Module Frequency Types

```
typedef enum _GFIX_MODULEFREQ_TYPE
 {
    GFIX_MODULEFREQ_OPTIMAL = 0,
    GFIX_MODULEFREQ_MINIMUM,
    GFIX_MODULEFREQ_MAXIMUM,
    GFIX_MODULEFREQ_REQUESTED,
    GFIX_MODULEFREQ_ACTUAL
} GFIX_MODULEFREQ_TYPE;
```

GFIX_GPIO_TYPE

Host **GPIO1–GPIO7** correspond to **HGP0–HGP6**, **C32KHZ**.

Host GPIO Types

```
typedef enum
{
    GFIX_GPIO0 = 0,
    GFIX_GPIO1,
    GFIX_GPIO2,
    GFIX_GPIO3,
    GFIX_GPIO4,
    GFIX_GPIO5,
    GFIX_GPIO6,
    GFIX_GPIO7
} GFIX_GPIO_TYPE;
```

GFRmIx Definitions

Module Operation Definitions

Module Operations

```
#define GFIX_DISABLECLOCK       0x00000001
#define GFIX_ENABLECLOCK        0x00000002
```

Module Operations

```
#define GFIX_SPECIAL            0x00000004
#define GFIX_FORCEDISABLECLOCK  0x00000008
#define GFIX_FORCEENABLECLOCK   0x00000010
#define GFIX_DYNAMICSWITCH      0x00000020
```

Clock Selections, Profiles, and Options

Clock Selections, Profile, and Options

```
#define GFIX_OSC            0x00000000
#define GFIX_ROSC           0x00000001
#define GFIX_PLL1           0x00000002
#define GFIX_PLL2           0x00000003
#define GFIX_REFCLK0        0x00000004
#define GFIX_REFCLK1        0x00000005
#define GFIX_PLL_QUERY      0x80000000
#define GFIX_POWER          0x00010000
#define GFIX_PERFORMANCE    0x00020000
#define GFIX_ALL_CLOCKS   ((1 << GFIX_OSC)|(1 << GFIX_ROSC)|
                           (1 << GFIX_PLL1)|\
                          (1 << GFIX_PLL2)|(1 << GFIX_REFCLK0)|
                          (1 << GFIX_REFCLK1))
#define GFIX_NO_PLLS        ((1 << GFIX_OSC)|(1 << GFIX_ROSC)|
                            (1 << GFIX_REFCLK0)|\
                            (1 << GFIX_REFCLK1))
#define GFIX_N_CLOCKSOURCES 6
#define GFIX_NO_CLOCK       0xFFFFFFFF
#define GFIX_DEFAULTSLK     0x00000000
  // Module chooses the closest frequency to the requested
#define GFIX_HIGHESTSLK     0x00000001
  // Module chooses highest frequency clock currently turned on
```

GPIO Operation Definitions

.

Host GPIO Operations

```
#define GFIX_GPIO_GET_INPUT_ENABLE                0x80000000UL
    // Get input enable.
#define GFIX_GPIO_GET_OUT_PUT_ENABLE              0x40000000UL
    // Get output enable.
#define GFIX_GPIO_SET_DATA                        0x20000000UL
    // Output data set to 1.
#define GFIX_GPIO_CLR_DATA                        0x10000000UL
    // Output data clear to 0.
#define GFIX_GPIO_GET_DATA                        0x08000000UL
    // Fills in_data of GFGPIOSTATUS.
#define GFIX_GPIO_SET_INPUT_ENABLE                0x04000000UL
    // Input enable set to 1.
#define GFIX_GPIO_CLR_INPUT_ENABLE                0x02000000UL
    // Input enable clear to 0.
#define GFIX_GPIO_SET_OUTPUT_ENABLE               0x01000000UL
   // Output enable set to 1.
#define GFIX_GPIO_CLR_OUTPUT_ENABLE               0x00800000UL
   // Output enable clear to 0.
```

GPIO Output Configuration Definitions

When configured as an output, it is necessary to select output as data, clocks, or interrupt.

GFIX_GPIO_CFG_OUTPUT can be ORed with **GFIX_GPIO_CFG_OUTPUT_SEL_***

```
/*
 * Configure the GPIO pin as data output.
 * Meaningful for pin0~pin4(HGP0~HGP4)
 */
#define GFIX_GPIO_CFG_OUTPUT_SEL_DATA             0x00000000
```

```
/*
 * Configure the GPIO pin as interrupt output.
 * Meaningful for pin3(HGP3) only
 */
#define GFIX_GPIO_CFG_OUTPUT_SEL_INTR             0x00000040
```

```
/*
 * GPIO output select RDY signal
 * Meaningful for pin4(HGP4) only
 */
#define GFIX_GPIO_CFG_OUTPUT_SEL_RDY             0x00000100

/*
 * For pin5 and pin6(HGP5, HGP6), config output clock or data
on HGP5 and HGP6.
 */
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP56_CLK       0x00000800
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP56_DATA      0x00000000

/*
 * For pin5(HGP5)
 * select monitor clock to HGP5
 */
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP5_PLL1       0x00000000
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP5_PLL2       0x00001000
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP5_PLLCOSC    0x00002000
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP5_PLLROSC    0x00003000

/*
 * For pin6(HGP6)
 * select monitor clock to HGP6
 */
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP6_PLL2       0x00000000
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP6_DCFCLK     0x00004000
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP6_MCCLK      0x00008000
#define GFIX_GPIO_CFG_OUTPUT_SEL_HGP6_DSPCLK     0x0000C000

/*
 * For pin7(C32KHZ)
 */
#define GFIX_GPIO_CFG_OUTPUT_SEL_C32KHZ          0x00000000
```

CHAPTER

4

Initialization API (GFIxAPI)

Overview

The initialization API (GFIxAPI) is the part of the GoForce Software Development Kit (GFSDK) that handles the initialization of GoForce media processors, including the clocks, clock source, memory interface unit (MIU), and I/O pad interface settings. The source code for the API can be licensed by OEMs (original equipment manufacturers) and ODMs (original design manufacturers).

The unified GFSDK supports separate initialization tables for the GoForce 4000, GoForce 4000 Rev B, GoForce 3D 4800, and GoForce 5000. At power up, the initialization code automatically detects the GoForce chip and initializes the chip from the appropriate table

Note to Application Developers

Although the entire API is callable by developers of applications for windowing and protected operating systems such as Windows CE, Palm OS, and Symbian, not all of the calls will have an effect. Some calls are made operational only for system and kernel developers who are integrating the GFSDK into kernel modules. Unlike other GoForce APIs, most of the GFIxAPI functions are not protected by kernel objects, such as semaphores and critical sections, because some GFIxAPI functions are called from protected code in other APIs.

GFIxAPI Tools and Environment

The standard GFIxAPI consists of source code, a sample demonstration application in source code form, and documentation. It is under the umbrella of the top-level GFSDK package.

The GFIxAPI was created using Microsoft Visual C^{++} 6.0. For systems using an NVIDIA PCI board running on Windows XP or Windows 2000, the GFSDK provides both the Visual C^{++} project environment and a debugging environment for the API, sample code, and programs. This Visual C^{++} environment can be a great help to any OEM who is developing and debugging software.

Note: PCI boards can be purchased through your local NVIDIA sales representative.

GFIxAPI Reference

The GFIxAPI includes a set of functions that initialize the NVIDIA platform media processor. It supports initialization of the device-configuration and other registers, the clock, the MIU, and I/O pad interface settings for Host, FP, VI, and SD. Please see the *GoForce Operating System Adaptation Guide*.

The GFIxAPI enables the frequencies for minimum power usage and maximum performance to be determined based on the API modules that are in use. It allows the appropriate frequencies to be set and sets other related parameters automatically.

For SC15, GFIxAPI functions are wrappers to the corresponding GFRmIx functions, as described in the following sections:

General GFIxAPI Functions

There is one general GFIxAPI function:

GFIxGetProperty()

Reference "GFRmIxGetProperty() and GFRmIxGetPropertyWithCS()" on page 62.

This function returns the fixed properties of the display hardware, such as the device ID, version, revision, and build number.

Function Prototype

```
GF_RETTYPE  GFIxGetProperty(
    GF_HANDLE      IxHandle,
    PGFIXPROPERTY  pIxProp );
```

Parameters

`IxHandle`	Handle obtained by using the resource manager's `GFRmComponentGet()` function.
`pIxProp`	Pointer to "GFIXPROPERTY Data Structure" on page 105.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

Host Interface and Clock-Related Functions

The host interface and clock-related GFIxAPI functions are as follows:

GFIxInit()

Reference "GFRmIxInit() and GFRmIxInitWithCS()" on page 64.

`GFIxInit()` initializes the I/O pad interface settings for Host, FP, VI, and SD. It also initializes clocks, clock sources, the MIU, device-control registers, and threshold values for different modules.

Function Prototype

```
GF_RETTYPE  GFIxInit(
    GF_HANDLE  IxHandle );
```

Parameters

`IxHandle`	Handle specific to the GFIxAPI.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIxDeInit()

Reference "GFRmIxDeInit() and GFRmIxDeInitWithCS()" on page 64.

`GFIxDeInit()` is the inverse of GFIxInit(), and shut downs the chip. All modules are reset and clocks are turned off.

Function Prototype

```
GF_RETTYPE  GFIxDeInit(
    GF_HANDLE  IxHandle );
```

Parameters

`IxHandle`	Handle specific to the GFIxAPI.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIxEnableClock()

Reference "GFRmIxEnableClock() and GFRmIxEnableClockWithCS()" on page 75.

`GFIxEnableClock()` allows turning on or off all the clocks to all GoForce processor modules for power savings. This function should be used by OEMs and ODMs (but not application developers) when the entire system enters power-saving mode.

Before disabling all clocks, the application should close all APIs except the GFIxAPI by calling **`GFRmComponentRelease()`**. When this function is called to enable clocks, all clocks are set to their default boot settings.

Function Prototype

```
GF_RETTYPE  GFIxEnableClock(
    GF_HANDLE  IxHandle,
    NvU32      option );
```

Parameters

`IxHandle`	Handle specific to the GFIxAPI.
`option`	

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIxEnablePLL()

This function enables or disables the phase-locked loop (PLL). It can be called multiple time by different APIs. If the request is to enable the PLL and the PLL is already enabled, the function returns without taking any action. If the request is to disable the PLL and it is already disabled, the function returns without taking any action.

If the PLL is to be *enabled*, the function does the following:

1. Enables the oscillator (the reference clock for the PLL).
2. Programs the PLL value.
3. Waits for the PLL to settle.
4. Switches the MIU to the PLL frequency.
5. Switches the bus interface unit (BIU) to the PLL frequency.

6. Switches the graphics engine to the PLL frequency.

If the PLL is enabled, this function also calls a platform-dependent function to switch to more efficient host bus timing for higher frequency PLL settings. It is assumed that the previous host bus timing settings were for a low-frequency relaxation oscillator or an external clock source. This option is available at compile time in source code form.

If the PLL is to be *disabled*, the function does the following:

1. Waits for graphics engine idle (GEIdle).
2. Waits for vertical sync if the display in enabled so the MIU is idle.
3. Switches the MIU frequency to the relaxation oscillator (ROSC) frequency.
4. Switches the BIU to the ROSC.
5. Switches the graphics engine to the ROSC.
6. Disables the PLL.
7. Turns off the reference clock.

If the PLL is disabled, this function also calls a platform-dependent function to switch to more relaxed host bus timing for a lower frequency relaxation oscillator or an external clock source. It is assumed that the previous host bus timing settings were for a high-frequency PLL clock source. This option is available at compile time in source code form.

Function Prototype

```
GF_RETTYPE  GFIxEnablePLL(
    GF_HANDLE  IxHandle,
    NvU32      option,
    NvU32      value );
```

Parameters

`IxHandle`	Handle specific to the GFIxAPI.
`option`	`GFIX_PLL1`.
`value`	0 for disable. 1 for enable.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIx3DReset()

Note: This function is called internally and is only for advanced use.

`GFIx3DReset()` resets the 3D engine.

Function Prototype

```
GF_RETTYPE  GFIx3DReset(
    GF_HANDLE  IxHandle );
```

Parameters

`IxHandle`	Handle specific to the GFIxAPI.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIxSetFrequency()

Reference "GFRmIxSetFrequency() and GFRmIxSetFrequencyWithCS()" on page 67.

This function sets a clock to a certain frequency. If the clock cannot meet the requested frequency, then the closest one will be chosen. For PLLs, this function determines the closest dividers to arrive at the requested frequency. To determine the frequency that will be chosen without actually setting it, specify GFIX_PLL_QUERY along with the clock.

Some clocks have only one frequency that never changes. In this case, the function must still be called at least once to allow the internal logic to know what the frequency of the clock is. To disable the selection of a clock by a module, a clock can be disabled by setting its frequency to zero; this will remove it from consideration for all modules.

This function will fail if a module is already using the clock.

Function Prototype

```
GF_RETTYPE  GFIxSetFrequency(
    GF_HANDLE  IxHandle,
```

Function Prototype (continued)

```
    NvU32       option,
    NvU32       *pFreq );
```

Parameters

IxHandle	Handle specific to the GFIxAPI.
option	GFIX_PLL1 or GFIX_PLL_QUERY
*pFreq	Pointer to 32-bit value in KHz to get three digit MHz precision. Returns with newly set current frequency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFIxGetFrequency()

Reference "GFRmIxGetFrequency() and GFRmIxGetFrequencyWithCS()" on page 68.

This function gets the value of the current PLL frequency, or the ROSC (relaxation oscillator clock) if the PLL is disabled. This function should be called by the API to compute the divisor for their module. .

Function Prototype

```
GF_RETTYPE  GFIxGetFrequency(
    GF_HANDLE  IxHandle,
    NvU32      option,
    NvU32      *pFrequency );
```

Parameters

IxHandle	Handle specific to the GFIxAPI.
option	GFIX_OSC, GFIX_ROSC, or GFIX_PLL1
*pFrequency	Pointer to 32-bit value in KHz to get three digit MHz precision. Returns with current frequency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFIxGPIO()

Reference "GFRmIxGPIO() and GFRmIxGPIOWithCS()" on page 78.

This function manipulates the GPIO pin signals.

Function Prototype

```
GF_RETTYPE  GFIxGPIO(
    GF_HANDLE          IxHandle,
    GFIX_GPIO_TYPE     gpio,
    NvU32              operation,
    PGFGPIOSTATUS      pGPIOStatus );
```

Parameters

IxHandle	Handle specific to the GFIxAPI.
gpio	One of the GFIX_GPIO_TYPE enumerated values. See"GFIX_GPIO_TYPE" on page 83.
operation	Combination of the operations described in "Host GPIO Operations" on page 85 and "GPIO Output Configuration Definitions" on page 85 .
pGPIOStatus	Pointer to the return status information. See "GFGPIOSTATUS" on page 16.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFIxEnableModuleClock()

Reference "GFRmIxEnableModuleClock() and GFRmIxEnableModuleClockWithCS()" on page 65.

This function enables and disables a module's clock as well as enables the module's source clock. There are two types of enables, the normal enable and the force enable. When doing a normal enable the module's clock enable reference count is increased as well as a call to **GFRmIxEnableClockSource()** which ensures the module's source clock is enabled. When doing a force enable/disable the modules reference counter is not touched, **GFRmIxEnableClockSource()** is called which increases/deceases the source's reference count. If the operation is to enable and the clock is currently disabled, for the both normal enable and force disable

GFRmIxEnableModuleClock will attempt to find the best clock source and divider for the module.

Function Prototype

```
GF_RETTYPE  GFIxEnableModuleClock(
    GF_HANDLE    IxHandle,
    eGFModuleIDs modid,
    NvU32        option,
    NvU32        special );
```

Parameters

IxHandle	Handle specific to the GFIxAPI
modid	ID of the module
option	Bit field of options, GFIX_ENABLECLOCK, GFIX_DISABLECLOCK, GFIX_FORCEENABLECLOCK, GFIX_FORCEDISABLECLOCK, GFIX_DYNAMICSWITCH, GFIX_SPECIAL
special	Specifies a module specific setting, only used when option contains GFIX_SPECIAL

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFIxEnableModule()

Reference "GFRmIxEnableModule() and GFRmIxEnableModuleWithCS()" on page 66.

This function takes a module out of reset. Each module has a reference count which is used to keep track of how many outstanding enables the module has. The module will not be disabled until the reference counter is returned to zero.

Function Prototype

```
GF_RETTYPE  GFIxEnableModule(
    GF_HANDLE    IxHandle,
    eGFModuleIDs modid,
    NvU32        option,
    NvU32        ComponentType,
    GF_HANDLE    ComponentHandle );
```

Parameters

`IxHandle`	Handle specific to the GFIx
`modid`	ID of the module
`option`	If option is nonzero the module will be enabled.
`ComponentType`	Type of component requesting this operation. Extra information included only to be passed to platform code.
`ComponentHandle`	Handle to component requesting this operation. Extra information included only to be passed to platform code.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIxSetModuleFrequency()

Reference "GFRmIxSetModuleFrequency() and GFRmIxSetModuleFrequencyWithCS()" on page 68.

This function sets the desired frequency of a module. The frequency must lie between the minimum and maximum frequencies specified in GFIxSetModuleConfig(). This value is cached and used when GFIxEnableModuleClock() is called attempting to enable the clock for the module.

Function Prototype

```
GF_RETTYPE  GFIxSetModuleFrequency(
    GF_HANDLE    IxHandle,
    eGFModuleIDs modid,
    NvU32        frequency );
```

Parameters

`IxHandle`	Handle specific to the GFIxAPI
`modid`	ID of the module.
`frequency`	32-bit value in KHz to get three digit MHz precision

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIxPowerPlane()

Reference "GFRmIxPowerPlane() and GFRmIxPowerPlaneWithCS()" on page 70.

This function enables and disables power plane(s). Multiple power planes can be disabled in one function call. There are two versions of this function specified by the compilation flag GF_EXTERNAL_POWERPLANE_LOGIC. If this flag is set to 1 then all the power plane logic is performed in the platform code. If set to 0 then all the logic is performed in Ix.

When internal power plane logic is chosen, each power plane has a reference count. If the power plane is to be enabled or disabled, GFIxPowerPlane() will call RmPowerPlane() to perform platform-specific enabling.

When external power plane logic is chosen, all calls to GFIxPowerPlane() are forwarded to the platform code, and the GFSDK does not perform

synchronization. The platform code is then responsible for reference counting and programming the registers.

Function Prototype

```
GF_RETTYPE  GFIxPowerPlane(
    GF_HANDLE                IxHandle,
    NvU32                    ppID,
    GFIX_POWERPLANE_OP_TYPE  operation,
    NvU32                   *states,
    NvU32                    ComponentType,
    GF_HANDLE                ComponentHandle,
    eGFModuleIDs             modid );
```

Parameters

IxHandle	Handle specific to the GFIxAPI
ppID	Bit field of power planes to be operated on
operation	Operation to perform on the power planes
*states	Ptr to 32 bit value to store the resulting bit field of a GFIX_POWERPLANE_OP_QUERY operation.
ComponentType	Type of component requesting this operation. Extra information included only to be passed to platform code
ComponentHandle	Handle to component requesting this operation. Extra information included only to be passed to platform code
modid	ID of the module requesting this operation. Extra information included only to be passed to platform code

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFIxEnableClockSource()

Reference "GFRmIxEnableClockSource() and GFRmIxEnableClockSourceWithCS()" on page 72.

This function enables and disables a clock's source. A reference counter is used to ensure the clock's source won't be disabled until there is a matching number of enable and disable operations. If a clock's source is to be enabled or disabled, **`GFIxEnableClockSource()`** will call **`RmEnableClockSource()`** to do the platform-specific enabling or disabling.

Function Prototype

```
GF_RETTYPE  GFIxEnableClockSource(
    GF_HANDLE IxHandle,
    NvU32     clockid,
    NvU32     state );
```

Parameters

`IxHandle`	Handle specific to the GFIx
`clockid`	ID of the clock
`state`	0 for disable 1 for enable

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIxSetPLLReferenceClocks()

Reference "GFRmIxSetPLLReferenceClocks() and GFRmIxSetPLLReferenceClocksWithCS()" on page 73.

This function sets the list of reference clocks that a PLL may choose.

Function Prototype

```
GF_RETTYPE  GFIxSetPLLReferenceClocks(
    GF_HANDLE IxHandle,
    NvU32     option,
    NvU32     clocks );
```

Parameters

`IxHandle`	Handle specific to the GFIxAPI
`option`	0 for disable 1 for enable.
`clocks`	Bit field of clocks. (1<<GFIX_OSC), (1<<GFIX_ROSC), (1<<GFIX_PLL1), (1<<GFIX_PLL2), (1<<GFIX_REFCLK1), (1<<GFIX_REFCLK2)

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFIxSetModuleConfig()

Reference "GFRmIxSetModuleConfig() and GFRmIxSetModuleConfigWithCS()" on page 74.

This function sets the bounds and clocking profile for a module. While this function may be called while the module's clock is enabled, it will not affect the current clock settings of the module, All new Ix calls dealing with the module will follow the configuration set by this function.

Function Prototype

```
GF_RETTYPE  GFIxSetModuleConfig(
    GF_HANDLE         IxHandle,
    eGFModuleIDs      modid,
    GFIXMODULECONFIG *pConfig );
```

Parameters

IxHandle	Handle specific to the GFIxAPI
modid	ID of the module
*pConfig	Pointer to the module configuration

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFIxGetModuleFrequency()

Reference "GFRmIxGetModuleFrequency() and GFRmIxGetModuleFrequencyWithCS()" on page 69.

This function retrieves clock frequency information about a module.

Function Prototype

```
GF_RETTYPE  GFIxGetModuleFrequency(
    GF_HANDLE              IxHandle,
    eGFModuleIDs           modid,
    GFIX_MODULEFREQ_TYPE   type,
    NvU32                 *pFrequency );
```

Parameters

IxHandle	Handle specific to the GFIxAPI
modid	ID of the module
type	Type of frequency
*pFrequency	Pointer to 32-bit value in KHz to get three-digit MHz precision. Returns with the current frequency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Attribute Functions

The attribute-related GFIxAPI functions are the following:

GFIxSetAttribute()

Reference "GFRmIxSetAttribute() and GFRmIxSetAttributeWithCS()" on page 79.

GFIxSetAttribute() sets the display attributes for software flag registers 1 and 2, and disables the **GFIxInit()** function.

Function Prototype

```
GF_RETTYPE  GFIxSetAttribute(
    GF_HANDLE       IxHandle,
    GFIX_ATTR_TYPE  attrType,
    NvU32           attrData, );
```

Parameters

IxHandle	Handle specific to the GFIxAPI.
attrType	Attribute type
attrData	Attribute data

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFIxGetAttribute()

Reference "GFRmIxGetAttribute() and GFRmIxGetAttributeWithCS()" on page 80.

This function gets the display attributes for software flag registers 1 and 2, and the enable/disable status of **GFIxInit()**.

Function Prototype

```
GF_RETTYPE  GFIxGetAttribute(
/* GF_RETTYPE  GFIxGetAttributeWithCS( */
    GF_HANDLE       IxHandle,
```

Function Prototype (continued)

```
    GFIX_ATTR_TYPE  attrType,
    NvU32           *pAttrData, );
```

Parameters

IxHandle	Handle specific to the GFIxAPI.
attrType	Attribute type
attrData	Attribute data

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFIXPROPERTY Data Structure

GFIXPROPERTY is the only data structure described in this section. It is used by "GFIxGetProperty()" on page 89.

GFIXPROPERTY Structure

```
typedef struct  _GFIXPROPERTY {
    NvU32  Version;
    NvU16  DeviceID;
    NvU16  DeviceRev;
    NvU32  BuildNumber;
    NvU32  Capability;
    NvU32  CapabilityEx;
} GFIXPROPERTY, *PGFIXPROPERTY;
```

GFIXPROPERTY Fields

Version	Version.
DeviceID	Device ID supported.
DeviceRev	Silicon revision supported.
BuildNumber	Build number.
Capability	Individual component's capability.
CapabilityEx	Individual component's extended capability .

GFIxAPI Attributes and Definitions

This section describes the attributes and definitions (those included in **`#define`** statements) that are part of the GFIxAPI. Abbreviations and acronyms found in the attribute and definition code are listed below.

GFIxAPI Abbreviations and Acronyms

ADDR	Address
B0	Buffer 0
B1	Buffer 1
BIU	Bus interface unit
BPP	Bits per pixel
BUF	Buffer
CLK	Clock
DBL	Double
DIS	Disable
EN	Enable
FP	Flat panel controller
FRC	Frame rate control
GC	Graphics controller
GE	Graphics engine
GEIdle	Graphics engine idle
GPIO	General purpose I/O
MIU	Memory interface unit
OSC	Oscillator
PLL	Phase-locked loop
POS	Position
PWM	Pulse width modulation
REG	Register
RLEX, RLX	Relaxation oscillator
ROSC	Relaxation oscillator
SC4	GoForce 4000
SC5	GoForce 4000 Rev B
SC10	GoForce 5000
SC12	GoForce 3D 4800

GFIxAPI Abbreviations and Acronyms (continued)

SD	Secure Digital Interface
SRAM	Static random access memory
SW	Software

GFIxAPI Attributes

These are the attributes used by the GFIxAPI. All of them should be set after calling **`GFIxInit()`**.

GFIXATTRIBUTES Enumeration Type

```
typedef enum  _GFIXATTRIBUTES {
    GFIX_ATTR_SW_REG_1,
        // Software register 1 for program usage.
    GFIX_ATTR_SW_REG_2,
        // Software register 2 for program usage.
    GFIX_ATTR_IX_INIT
        // Set GFIxInit() to on or off.
} GFIXATTRIBUTES;
```

Initial Programming Sequence

The best way to learn and use the GFIxAPI is to refer to the sample application source code that comes with the GFSDK package. True beginners might start a new application by first creating one of their own at the same directory level as the Demo (or Samples) application source directory. This way the same directory tree structure can be maintained. The general sequence for programming the GFIxAPI follows.

1. Set the main panel type (and the subpanel type if subpanel support is enabled) and other configuration flags in GFPlat.h or GFExtCfg.h, and compile the required project.
2. Display application should call **`GFRmOpen()`** to get a resource manager handle to resource services like memory surface allocation, and register and memory access.
3. The application should call the **`GFRmComponentGet()`** function to get an **`IxHandle`** to access the GFIxAPI and to initialize the GFIxAPI function call pointers.
4. If multiple panel support is enabled, use **`GFIxSetAttribute()`** to set the main panel type (and the subpanel type if subpanel support is enabled). This function should be called twice if a subpanel is used.
5. Calling the **`GFIxGetProperty()`** function returns device information and main LCD and sub-LCD size information.
6. The application should call **`GFIxInit()`** to initialize clock sources, clocks (PLL), the oscillator for the PLL and relaxation oscillator, thresholds, and SRAM parameters.
7. **`GFRmComponentRelease()`** frees the display component handle, **`IxHandle`**.
8. **`GFRmClose()`** closes the resource manager. All necessary cleaning up is performed here. The application is required to call this function.

CHAPTER

5

Interrupt Architecture API (GFINTxAPI)

Interrupt Handling Overview

Because interrupt support is highly platform dependent, NVIDIA GoForce hardware interrupt handling involves the two-tiered architecture that is described below:

- "System-Level Interrupt Control" on page 109
- "Component-Level Interrupt Control" on page 110

The GoForce software development kit (GFSDK) exposes these hardware interrupts to software through two software implementations:

- The GFINTxAPI component, which is described in this chapter.
- Sample applications that follow an interrupt service routine (ISR) and interrupt service thread (IST) programming model. ISRs are usually in the operating system kernel. ISTs could be user level.

System-Level Interrupt Control

System-level interrupt control is highly dependent on the platform and operating system. Most of the operating system environments that the GFSDK operates in have a set of global interrupt routines that is closely

related to the interrupt mechanism of a particular CPU. Applications should always make use of these global interrupt routines. Under an ISR/IST interrupt model, this could mean a system builder needs to modify the ISR of the host CPU. A detailed description of implementing this level of interrupt control in an ISR is found in a separate document, the *GoForce Media Processor Platform Adaptation Guide.*

The functions for controlling system-level interrupts listed in this chapter are mainly for use by system builders as a reference. The functions are not commonly used in user-level applications, especially for platforms and operating systems that have strict kernel and user-space separation.

Component-Level Interrupt Control

Component-level interrupt support is achieved by calling interrupt-enable, interrupt-clear, interrupt-disable, and interrupt-handler functions for each component. These are component-level interrupts that are usually handled by the individual GFSDK API components.

The interrupt handler in a GFSDK component fulfills the corresponding interrupt task and is transparent to application developers. The interrupt handler is unique to each GFSDK component; it is advisable to review the detailed description of it that is found in the documentation for the component.

GFINTxAPI Operating System Dependency

The GFINTxAPI is also dependent on an operating system's interrupt handling. This allows GFINTxAPI functions to be any of the following:

- Directly called functions
- Wrappers for the operating system's interrupt-related functions
- Implementation references in interrupt service routines

One GFINTxAPI sample implementation is provided for the Windows CE operating system and includes references for a Windows CE kernel ISR implementation. All GFINTxAPI functions listed in this chapter are simply wrappers to for Windows CE functions.

GFINTxAPI Reference

The GFINTxAPI reference contains the "GFINTxAPI Functions" on page 111 and the "GFINTxAPI Data Types" on page 120.

GFINTxAPI Functions

The NVIDIA GoForce Interrupt API provides a complete set of interrupt routines for OEMs to use. The functions are described below.

GFINTxPinInitialize()

`GFINTxPinInitialize()` initializes the `INTRn` pin and enables interrupt generation. It's use is not recommended for OEMs that maintain a kernel/user context because its function is handled at the kernel level.

Function Prototype

```
GF_RETTYPE  GFINTxPinInitialize(
    GF_HANDLE  INTxHandle,
    NvU32      cfgData );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
cfgData	Possible value combinations are defined in GFSCIReg.h. As an example, configuring the INTRn pin as an interrupt pin with INTRn polarity active low would look like this: `cfgData = INTRn_PIN_INT | INTRn_ACTV_L;`

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxPinStatus()

This routine queries for the current INTRn pin configuration, whose value is pointed to by **pCfgData**. See **GFINTxPinInitialize()**, above. It's use is not recommended for OEMs that maintain a kernel/user context because its function is handled at the kernel level.

Function Prototype

```
GF_RETTYPE  GFINTxPinStatus(
    GF_HANDLE  INTxHandle,
    NvU32      *pCfgData );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
*pCfgData	Pointer to the configuration value.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxInitialize()

This function enables the system interrupt that has interrupt ID **idInt**. It can serve as an operating system function wrapper or be part of an OEM-specific

interrupt scheme. Its current configuration, which is for Windows CE, is as operating system wrapper.

Function Prototype

```
GF_RETTYPE  GFINTxInitialize(
    GF_HANDLE  INTxHandle,
    NvU32      idInt,
    GF_HANDLE  hEvent,
    void       *pData,
    NvU32      DataSize );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
idInt	System interrupt ID. This value is operating system dependent.
hEvent	Handle to an event.
*pData	Pointer to the input data.
DataSize	Size of the data.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxClear()

This routine clears the interrupt denoted by **idInt**. It is not recommended for use by OEMs that maintain a kernel/user context.

Function Prototype

```
GF_RETTYPE  GFINTxClear(
    GF_HANDLE  INTxHandle,
    NvU32      idInt );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
idInt	System interrupt ID. This value is operating system dependent.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxSetStatusCtrl()

This function enables and disables interrupt generation. It is not recommended for use by OEMs that maintain a kernel/user context.

Function Prototype

```
GF_RETTYPE  GFINTxSetStatusCtrl(
    GF_HANDLE  INTxHandle,
    NvU32      fDisable );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
fDisable	If this value is true, interrupt generation is disabled. If false, interrupt generation is enabled.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxDisable()

GFINTxDisable() disables the interrupt denoted by **idInt**. It can serve as an operating system function wrapper or be part of an OEM-specific interrupt scheme. Its current configuration, which is for Windows CE, is as operating system wrapper.

Function Prototype

```
GF_RETTYPE  GFINTxDisable(
    GF_HANDLE  INTxHandle,
    NvU32       idInt );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
idInt	System interrupt ID. This value is operating system dependent.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxEnable()

This function enables the interrupt denoted by **idInt**. It is not recommended for use by OEMs that maintain a kernel/user context.

Function Prototype

```
GF_RETTYPE  GFINTxEnable(
    GF_HANDLE  INTxHandle,
    NvU32      idInt );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
idInt	System interrupt ID. This value is operating system dependent.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxGetMask()

This function gets the interrupt mask denoted by **idInt**.

Function Prototype

```
GF_RETTYPE  GFINTxGetMask(
    GF_HANDLE            INTxHandle,
    NvU32                idInt,
    INTERRUPTMASKSTATUS  *pIntMaskStatus );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
idInt	System interrupt ID. This value is operating system dependent.
pIntMaskStatus	Returns the interrupt mask. The value is defined in the "INTERRUPTMASKSTATUS Enumeration Type" on page 120.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxGetStatus()

This function gets the interrupt status denoted by **idInt**.

Function Prototype

```
GF_RETTYPE  GFINTxGetStatus(
    GF_HANDLE         INTxHandle,
    NvU32             idInt,
    INTERRUPTSTATUS   *fStatus );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
idInt	System interrupt ID. This value is operating system dependent.
fStatus	Returns interrupt status. The value is defined in the "INTERRUPTMASKSTATUS Enumeration Type" on page 120.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxIntrMapping()

GFINTxIntrMapping() manages the mapping of system, logic, and hardware interrupts.

- **System interrupts** are unique to each operating system and OEMs must define their own that reflects the platform and system being used. Definitions related to system interrupts are included in the file **OEMSysIntr.h**.
- **Logic interrupts** are defined in the file **OEMIntrMapping.inl**, which OEMs should change in accordance with their platforms.
- **Hardware interrupts** corresponding NVIDIA GoForce hardware interrupt masks are defined in **GFSCHwrIntr.h**. OEMs can use the file for reference, but are not supposed to modify it.

OEMs are responsible for setting the mappings in **OEMIntrMapping.inl** for system, logic, and hardware interrupts. This is accomplished using the

predefined macro **SETUP_INTERRUPT_MAP**, which assigns a logic interrupt value that provides the NVIDIA GoForce hardware with a fixed interrupt priority. The lower the logic interrupt value, the higher the interrupt priority. OEMs should ensure that each entry in the interrupt table has a unique logic interrupt value and that the values are consistent.

The example below uses **SETUP_INTERRUPT_MAP** to map logic interrupt value 0 (the highest interrupt priority) to the hardware MPEG-4 encoder interrupt (MEI) and the system interrupt **SYSINTR_NV_MEI**.

```
SETUP_INTERRUPT_MAP( LOGICINT0, HWRINT_NV_MEI, SYSINTR_NV_MEI,
    NVSCIRQHANDLER_UNDEFINED );
```

NVSCIRQHANDLER_UNDEFINED is always used as the last parameter of **SETUP_INTERRUPT_MAP** and is reserved for future system environments.

Function Prototype

```
GF_RETTYPE  GFINTxIntrMapping(
    NvU32           idInt,
    NvU32           *pData,
    GFSCINT_OPTION  option );
```

Parameters

idInt	System interrupt ID. This value is operating system dependent.
*pData	Data to pass into or out of this function.
Option	Values are defined in "GFSCINT_OPTION Enumeration Type" on page 120.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxWaitForIRQ

Function Prototype

```
GF_RETTYPE  GFINTxWaitForIRQ(
    GF_HANDLE INTxHandle,
    NvU32 idInt);
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
idInt	System interrupt ID. This value is operating system dependent.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxHostSysToChVector

Function Prototype

```
GF_RETTYPE  GFINTxSysToChVector(
    GF_HANDLE INTxHandle,
    NvU32     sys,
    NvU32    *vector,
    NvU32    *ch);
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
sys	
*vector	
*ch	

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxHostChVectorToSys

Function Prototype

```
GF_RETTYPE  GFINTxHostChVectorToSys(
    GF_HANDLE INTxHandle,
    NvU32     ch,
    NvU32     vector,
    NvU32    *sys);
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
ch	
vector	
*sys	

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxHostReadVector()

Function Prototype

```
GF_RETTYPE  GFINTxHostReadVector(
    GF_HANDLE INTxHandle,
    NvU32     ch,
    NvU32     *vector);
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
ch	
*vector	

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxHostControl

Function Prototype

```
GF_RETTYPE  GFINTxHostControl(
    GF_HANDLE                    INTxHandle,
    GF_INTX_HOST_OPERATION_TYPE  op,
    NvU32                        idInt,
    void                        *pData );
```

Parameters

INTxHandle	Obtained by calling "GFRmComponentGet()" on page 26.
op	
idInt	System interrupt ID. This value is operating system dependent.
*pData	Data to pass into or out of this function.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFINTxAPI Data Types

There are three enumerated data types in the GFINTxAPI, which are represented by the variables **GFSCINT_OPTION**, **INTERRUPTSTATUS**, and **INTERRUPTMASKSTATUS**.

GFSCINT_OPTION Enumeration Type

```
typedef enum {
    GFSCINT_INITIALIZE,     // Interrupt table initialize.
    GFSCINT_GET_LOGIC,      // From SysIntr to LogicIntr.
    GFSCINT_GET_HWRIRQ,     // From LogicIntr to HwrIntr.
    GFSCINT_GET_SYSINTR,    // From LogicIntr to SysIntr.
    GFSCINT_GET_IRQHANDLER  // idInt is a logic intr.
} GFSCINT_OPTION;
```

INTERRUPTSTATUS Enumeration Type

```
typedef enum {
    INTERRUPT_STATUS_FALSE,
    INTERRUPT_STATUS_TRUE
} INTERRUPTSTATUS;
```

INTERRUPTMASKSTATUS Enumeration Type

```
typedef enum {
    INTERRUPT_MASK_STATUS_ON,   // Interrupt is disabled.
    INTERRUPT_MASK_STATUS_OFF   // Interrupt is enabled.
} INTERRUPTMASKSTATUS;
```

Interrupt Programming Assistance

This section presents a general description of how an application can use the NVIDIA hardware interrupt architecture. It assumes the platform's operating system follows an ISR or IST interrupt handling model. A concrete example of this procedure can be found under "MPEG-4 Encoder Interrupt Programming Assistance" on page 446.

1. The application creates an operating system event for the interrupt.
2. The application calls a GFSDK component API function, such as is found in the "MPEG-4 Encoder Interrupt API " on page 443, to enable the component-level interrupt.
3. The application calls a GFSDK GFINTxAPI function to enable the system-level interrupt.
4. In most cases, an operating system thread (an IST) should be created that is intended to handle this interrupt. The thread should be kept sleeping, waiting for the interrupt to occur.
5. When the interrupt does occur, the waiting thread is awakened by the operating system. The thread should complete its intended operations by calling the appropriate functions in the GFSDK component API.
6. Once the intended operation is completed, a component API function is called to clear the component-level interrupt.
7. Next, the GFSDK GFINTxAPI API function to clear the system-level interrupt is called.
8. Finally, the operating system event that was created is released.

The application may call the GFSDK component API that handles the interrupt operation multiple times to complete a interrupt request. One example is the MPEG-4 encoding operation where multiple calls to GFMxEncAPI API functions are made to obtain an entire encoded frame.

CHAPTER

6

Display API (GFDxAPI)

Overview

The display API (GFDxAPI) is the part of the GoForce Software Development Kit (GFSDK) that handles the initialization of GoForce media processors and LCD panels. The source code for the API can be licensed by OEMs (original equipment manufacturers) and ODMs (original design manufacturers).

Note to Application Developers

Although the entire API is callable by developers of applications for windowing and protected operating systems such as Windows CE, Palm OS, and Symbian, not all of the calls will have an effect. Some calls are made operational only for system and kernel developers who are integrating the GFSDK into kernel modules. Unlike other GoForce APIs, most of the GFDxAPI functions are not protected by kernel objects, such as semaphores and critical sections, because some GFDxAPI functions are called from protected code in other APIs.

GFDxAPI Tools and Environment

The standard GFDxAPI consists of source code, a sample demonstration application, and documentation. It is under the umbrella of the top-level

GFSDK package. The demonstration application is provided in source code form.

The GFDxAPI and the sample code (DxTest.c) were created using Microsoft Visual C++ 6.0. For systems using an NVIDIA PCI board running on Windows XP, Windows 2000, or Windows NT 4.0, the GFSDK provides both the Visual C++ project environment and an excellent debugging environment for the API, sample code, and programs. This Visual C++ environment can be a great help to any OEM who is developing and debugging software. The GFSDK also supports Cogent ARM7 and MQ9000 (ARM9-based) platforms using CodeWarrior ADS V1.1 (or greater) tools for development and code debugging.

Note: PCI boards can be purchased through your local NVIDIA sales representative.

GFDxAPI Reference

The GFDxAPI includes a set of functions that initialize the NVIDIA platform media processor. It supports initialization of the device configuration and other registers, the clock, the memory interface unit (MIU), and the color depth. It also initializes the NVIDIA media processor for a given main panel and subpanel (the subpanel is selected, if needed). The GFSDK also provides sample code to initialize the NVIDIA media processor for a given panel type at a 16-bpp color depth.

In order to support multiple LCD panels and to ease the selection of various hardware configurations, most of the GFDxAPI is table driven. The GFDxAPI also supports sub-LCD tables, which include the Basic Initialization Table, the Graphics Controller (GC) Timing Table, the Flat Panel (FP) Control Table, the Frame Rate Control (FRC) Table (for an STN LCD), the Power-on Sequence Table, and the Power-off Sequence Table. The main and sub-LCD tables are fixed and should be placed in the ROM. For more details, please see the *GoForce Operating System Adaptation Guide*.

The GFDxAPI supports different types of main panels and subpanels, which are identified through a parameter table based on panel specifications provided by OEMs and panel vendors. OEMs can compile for one main panel

or one main and one subpanel. If multiple panel support is enabled using the **GF_MULTI_PANEL** flag during compilation, OEMs can compile for multiple main panels or multiple main and subpanels. Multiple panel support allows **GFDxSetAttribute()** (see "GFDxSetAttribute() and GFDxSetAttributeWithCS()" on page 129) to set the one main panel or one main and one subpanel type. Additionally, secondary versions of **GFDxGetAttribute()** and **GFDxSetAttribute()** protect the primary versions by offering thread safe execution. The secondary versions have **WithCS** appended to the function name.

The GFDxAPI enables the frequencies for minimum power usage and maximum performance to be determined based on the API modules that are in use. It allows the appropriate frequencies to be set and sets other related parameters automatically.

Additionally, the GFDxAPI supports functions that erase screen memory and set the LCD color depth, the LCD stride, the graphics media processor rotation (to 0, 90, 180, or 270 degrees clockwise), and the power-up and power-down panel power sequencing.

The GFDxAPI is described in the following sections:

Note: Many of the GFDxAPI functions have counterparts in the GFIxAPI. In these cases, the GFDxAPI function is obsolete and is included for compatibility only.

General GFDxAPI Functions

There are four general GFDxAPI functions:

GFDxOpen()

This function is implicitly called by the resource manager (GFRm) in response to an application's call to **`GFRmComponentGet()`** for the GFDxAPI component. It is not supposed to be called by an application. It is included here only for the purpose of GFDxAPI source code customization.

`GFDxOpen()` is called by the GFRm when the **`GFRmComponentGet()`** service is called with **`GF_DXAPI`**. It is listed here only for the purpose of GFDxAPI source code customization.

Function Prototype

```
GF_HANDLE  GFDxOpen(
    GF_HANDLE  RmHandle );
```

Parameters

`RmHandle`	Resource manager handle.

Return Values. **`GFDxOpen()`** returns a handle specific to the GFDxAPI if successful and NULL if there is an error.

GFDxClose()

This function is implicitly called by the resource manager (GFRm) in response to an application's call to **`GFRmComponentGet()`** for the GFDxAPI component. It is included here only for the purpose of GFDxAPI source code customization.

Function Prototype

```
GF_RETTYPE  GFDxClose(
    GF_HANDLE  *pDxHandle );
```

Parameters

`*pDxHandle`	Pointer to display handle.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFDxGetProperty()

This function returns the fixed properties of the display hardware, such as the main LCD display size and the sub-LCD display size.

Function Prototype

```
GF_RETTYPE  GFDxGetProperty(
    GF_HANDLE      DxHandle,
    PGFDXPROPERTY  pDxProp );
```

Parameters

`DxHandle`	Handle obtained by using the resource manager's `GFRmComponentGet()` function.
`pDxProp`	Pointer to `GFDXPROPERTY` structure. See "GFDXPROPERTY" on page 133.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

Display-Related Functions

The display-related GFDxAPI functions are the following:

GFDxSetDisplay() and GFDxSetDisplayWithCS()

This function initializes the main LCD or sub-LCD display. All the GFDxAPI attributes should be set after the **`GFDxSetDisplay()`** function call except for **`GFDX_ATTR_LCDTYPE`**, which should be set before the **`GFDxInit()`** function call to determine the main panel type (or the main and subpanel types if compiled with **`GF_MULTI_PANEL`** flag).

GFDxSetDisplayWithCS() has the same parameters as **GFDxSetDisplay()** and is provided for thread-safe calls. It has built-in, mutually exclusive code protection.

Function Prototype

```
GF_RETTYPE  GFDxSetDisplay(
/* GF_RETTYPE  GFDxSetDisplayWithCS( */
    GF_HANDLE  DxHandle,
    int        lcdSel,
    NvU32      ulStartAddr,
    NvU32      ulStride,
    unsigned   bpp );
```

Parameters

DxHandle	Handle specific to the GFDxAPI.
lcdSel	0 for LCD_MAIN. 1 for LCD_SUB.
ulStartAddr	Default display start address for main or sub-LCD (should be set to 0). GFDxSetAttribute() should be called to set the display start address for the main LCD or sub-LCD.
ulStride	Default stride if 0; otherwise use the passed value. GFDxSetAttribute() should be called to set the display stride for the main LCD or sub-LCD.
bpp	Default should be 16 bits/pixel for main LCD or sub-LCD. GFDxSetAttribute() should be called to set the display bpp for the main LCD or sub-LCD.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFDxEnableDisplay()

This function allows the turning on or turning off of a main display or subdisplay with a power sequence.

Function Prototype

```
GF_RETTYPE  GFDxEnableDisplay(
    GF_HANDLE  DxHandle,
```

Function Prototype (continued)

```
    int         lcdSel,
    unsigned    bOn );
```

Parameters

DxHandle	Handle specific to the GFDxAPI.
lcdSel	0 for LCD_MAIN. 1 for LCD_SUB.
bOn	0 to disable main or subdisplay. 1 to enable main or subdisplay.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFDxSetAttribute() and GFDxSetAttributeWithCS()

GFDxSetAttribute() sets the display attributes for a main panel or subpanel. All the GFDxAPI attributes should be set after the **GFDxSetDisplay()** function call except for **GFDX_ATTR_LCDTYPE**, which should be set before the **GFDxInit()** function call to determine the main panel type (or the main and subpanel types if compiled with the **GF_MULTI_PANEL** flag).

GFDxSetAttributeWithCS() has the same parameters as **GFDxSetAttribute()** and is provided for thread-safe calls. It has built-in, mutually exclusive code protection.

Function Prototype

```
GF_RETTYPE  GFDxSetAttribute(
/* GF_RETTYPE  GFDxSetAttributeWithCS( */
    GF_HANDLE  DxHandle,
    int        lcdSel,
    unsigned   aid,
    NvU32      attr );
```

Parameters

DxHandle	Handle specific to the GFDxAPI.
lcdSel	0 for LCD_MAIN. 1 for LCD_SUB.
aid	Attribute IDs. See "GFDXATTRIBUTES Enumeration Type" on page 138.
attr	Attribute value for main LCD or sub-LCD. Should be written with attribute ID in aid.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFDxGetAttribute() and GFDxGetAttributeWithCS()

GFDxGetAttribute() returns the display attributes for a main panel or subpanel. **GFDxGetAttributeWithCS()** makes a similar query but provides resource protection. Both functions have the same parameters.

Function Prototype

```
GF_RETTYPE  GFDxGetAttribute(
/* GF_RETTYPE  GFDxGetAttributeWithCS( */
    GF_HANDLE  DxHandle,
    int        lcdSel,
    unsigned   aid,
    NvU32      *attr );
```

Parameters

DxHandle	Handle specific to the GFDxAPI.
lcdSel	0 for LCD_MAIN. 1 for LCD_SUB.
aid	Attribute IDs. See "GFDXATTRIBUTES Enumeration Type" on page 138.
*attr	Attribute value for main LCD or sub-LCD. Is written with attribute ID in aid.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFDxErase() and GFDxEraseWithCS()

This function clears the main or subdisplay screen with the given color. It also clears the display buffer of the main or subpanel window. Window A buffer 0 is used as a display buffer.

GFDxEraseWithCS() has the same parameters as **GFDxErase()** and is provided for thread-safe calls. It has built-in, mutually exclusive code protection

Function Prototype

```
GF_RETTYPE  GFDxErase(
/* GF_RETTYPE  GFDxEraseWithCS( */
    GF_HANDLE  DxHandle,
    int        lcdSel,
    NvU32      ulStartAddr,
    NvU32      color );
```

Parameters

DxHandle	Handle specific to the GFDxAPI.
lcdSel	0 for LCD_MAIN. 1 for LCD_SUB.
ulStartAddr	Start address. (The offset in bytes of the display frame buffer.)
color	Clear screen with this color. The 32-bit value has a 16-bit color duplicated in the upper and lower parts.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFDxSwitchLCD()

This function enables switching between the main LCD and sub-LCD.

Function Prototype

```
GF_RETTYPE  GFDxSwitchLCD(
    GF_HANDLE  DxHandle,
    int        lcdSel,
    int        Option );
```

Parameters

DxHandle	Handle specific to the GFDxAPI.
lcdSel	0 for LCD_MAIN. 1 for LCD_SUB.
Option	0 for switch (Must be set to 0.)

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFDxVSync()

These functions wait for a vertical sync and return to the caller. **GFDxVSyncWithCS()** additionally provides protected hardware access.

Function Prototype

```
GF_RETTYPE  GFDxVSync(
    GF_HANDLE  DxHandle,
    NvU16      option );
```

Parameters

DxHandle	Handle specific to the GFDxAPI.
option	• GFDX_VSYNC_WAIT. Waits until vertical blank (default). • GFDX_VSYNC_CHECK. Checks and immediately returns vertical blank status. Returns GF_SUCCESS if in vertical blank, else returns GF_ERROR. • GFDX_VSYNC_HW_WAIT. Puts Wait token in the command FIFO and the CPU returns immediately

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFDxAPI Data Structures

The following data structures are described in this section:

GFDXPROPERTY

Used by **GFDxGetProperty()**.

GFDXPROPERTY Structure

```
typedef struct  _GFDXPROPERTY {
    GFPROPERTY  Prop;
    NvU16       LcdWidth;
    NvU16       LcdHeight;
    NvU16       LcdIndex;
    char        LcdName[18];
    NvU16       SubLcdWidth;
    NvU16       SubLcdHeight;
    NvU16       SubLcdIndex;
    char        SubLcdName[18];
} GFDXPROPERTY, *PGFDXPROPERTY;
```

GFDXPROPERTY Fields

Prop	Standard GFPROPERTY. See "GFPROPERTY" on page 10.
The following fields are available as capabilities.	
LcdWidth	Width of the panel in pixels.
LcdHeight	Height of the panel in lines.
LcdIndex	NVIDIA panel index in the database.
LcdName[18]	Panel name or description (up to 17 characters); NULL terminated.
SubLcdWidth	Width of the panel in pixels.
SubLcdHeight	Height of the panel in lines.

GFDXPROPERTY Fields (continued)

SubLcdIndex	NVIDIA panel index in the database.
SubLcdName[18]	Panel name or description (up to 17 characters); NULL terminated.

GFLCDCONFIG

This is the structure for the display parameter table. This is an internal structure used for GFDxAPI source code customization only.

GFLCDCONFIG Structure

```
typedef struct  _GFLcdConfig {
    NvU8      ucSignature[GF_SIGNATURE_LEN];
    NvU16     usDeviceId;
    NvU16     usRevision;
    NvU16     usSize;
    NvU16     usVersion;
    NvU8      ucTargetCPU;
    NvU8      ucTargetPlatform;
    NvU16     usPanelX;
    NvU16     usPanelY;
    NvU8      ucPanelBpp;
    NvU8      ucPanelRefr;
    NvU16     usPanelVX;
    NvU16     usPanelVY;
    NvU16     usPanelXA;
    NvU16     usPanelYA;
    NvU16     usPanelStride;
    NvU8      ucPanelType;
    NvU8      ucReserved;
    NvU32     ulFlags;
    NvU32     ulResetLcd;
    NvU32     ullcdIdxId;
    NvU32     ullcdDataId;
    NvU32     ulPerodicXfer;
    NvU32     ulRelxDiv;
    NvU32     ulExtOscDiv;
    NvU32     ulReserved3;
    NvU32     ulReserved4;
```

GFLCDCONFIG Structure (continued)

```
    NvU32     ulReserved5;
    NvU32     ulReserved6;
    NvU32     ulNGCTiming;
    REGENTRY  regGCTiming[N_GC_REGS];
    NvU32     ulNFPControl;
    REGENTRY  regFPControl[N_FP_REGS];
    NvU32     ulNPowerOn;
    REGENTRY  regPowerOn[N_POWERON_REGS];
    NvU32     ulNPowerOff;
    REGENTRY  regPowerOff[N_POWEROFF_REGS];
} GFLCDCONFIG, *PGFLCDCONFIG;
```

GFLCDCONFIG Fields

`ucSignature`	Signature to be searched in DLL file
`usDeviceId`	Device ID of the product, supported by this driver
`usRevision`	Revision of the processor
`usSize`	Size of the whole structure (including signature)
`usVersion`	Version number
`ucTargetCPU`	CPU (see index)
`ucTargetPlatform`	Platform (see index)
`usPanelX`	Panel Size X
`usPanelY`	Panel Size Y
`ucPanelBpp`	Panel bits per pixel
`ucPanelRefr`	Panel refresh rate in Hz
`usPanelVX`	Panel viewport width (can be smaller)
`usPanelVY`	Panel viewport height
`usPanelXA`	XA start offset
`usPanelYA`	YA start offset
`usPanelStride`	Panel stride in bytes (0 for default)
`ucPanelType`	Panel index (type) number
`ucReserved`	Filler for alignment
`ulFlags`	(*Reserved*)
`ulResetLcd`	LCD reset

GFLCDCONFIG Fields (continued)

`ullcdIdxId`	LCD driver index ID
`ullcdDataId`	LCD driver data ID
`ulPerodicXfer`	Send periodic command and data to LCD driver
`ulRelxDiv`	ROSC clock divisor for GC
`ulExtOscDiv`	External oscillator clock divisor for GC
`ulReserved3`	(*Reserved*)
`ulReserved4`	(*Reserved*)
`ulReserved5`	(*Reserved*)
`ulReserved6`	(*Reserved*)
`ulNGCTiming`	Number of parameters in GC table
`ulNFPControl`	Number of parameters in FP table
`ulNPowerOn`	Power-up sequence
`ulNPowerOff`	Power-down sequence

GFDxAPI Attributes and Definitions

This section describes the attributes and definitions (those included in **`#define`** statements) that are part of the GFDxAPI. Abbreviations and acronyms found in the attribute and definition code are listed below.

GFDxAPI Abbreviations and Acronyms

ADDR	Address
B0	Buffer 0
B1	Buffer 1
BIU	Bus interface unit
BPP	Bits per pixel
BUF	Buffer
CLK	Clock
DBL	Double
DIS	Disable
EN	Enable
FP	Flat panel controller

GFDxAPI Abbreviations and Acronyms (continued)

FRC	Frame rate control
GC	Graphics controller
GE	Graphics engine
GEIdle	Graphics engine idle
GPIO	General purpose I/O
MIU	Memory interface unit
OSC	Oscillator
PLL	Phase-locked loop
POS	Position
PWM	Pulse width modulation
REG	Register
RLEX, RLX	Relaxation oscillator
ROSC	Relaxation oscillator
SRAM	Static random access memory
SW	Software
VCLK	Video clock
VI	Video interface
WA	Window A
WB	Window B
WENC	Encode window
WH	Width and height

GFDxAPI Attributes

These are the attributes used by the GFDxAPI. All of them should be set after calling **`GFDxSetDisplay()`** except for **`GFDX_ATTR_LCDTYPE`**, which should be set before calling **`GFDxInit()`** to determine the main panel type (or the main and subpanel types if compilation was with the **`GF_MULTI_PANEL`** flag).

`GFDX_ATTR_OVERLAP_TYPE` is one attribute that requires explanation. When window A and window B overlap, it allows the selection of the color key from window A or window B by specifying the generation of the *overlay*

key. The overlay key determines whether the pixel data of window A or window B is displayed in the overlap area. The values are as follows:

00. Window A color key is generated in the overlap area when the window A pixel color (as found in the color palette) matches the specified color in the color key register. Setting the corresponding bits in the color key mask register masks individual bits in the color key comparator. When the key is generated, the corresponding window B pixels are visible.

01. Window B color key is generated in the overlap area when the window B pixel color (as found in the color palette) matches the specified color in the color key register. Setting the corresponding bits in the color key mask register masks individual bits in the color key comparator. When the key is generated, the corresponding window A pixels are visible.

GFDXATTRIBUTES Enumeration Type

```
typedef enum  _GFDXATTRIBUTES {
    GFDX_ATTR_LCDWIDTH,
        // RO width of the LCD fixed (240 for a 240x320 panel).
    GFDX_ATTR_LCDHEIGHT,
       // RO height of the LCD fixed (320 for a 240x320 panel).
    GFDX_ATTR_LCDTYPE,
        // RO NVIDIA panel index (2 for Philips 240x320 panel).
    GFDX_ATTR_LCDNAME,
       // RO Name of panel (up to 16 chars, "PHILIPS_LP7123A").
    GFDX_ATTR_WIDTH,
        // Width of the display area usually same as LCDWIDTH.
    GFDX_ATTR_HEIGHT,
        // Height of display area.
        // Can be less if emulating smaller size.
    GFDX_ATTR_WH_WA,
        // Set/gets width height together to avoid artifacts.
    GFDX_ATTR_XY_POS_WA,
        // Set/gets x, y start position of window A.
    GFDX_ATTR_BPP_WA,
        // Current bits/pixel: 8, 15(RGB:555), 16(RGB:565),
        // 18(RGB:LSB), 24(RGB:8888)
    GFDX_ATTR_PIXEL_DBL_WA,
        // On/off horizontal and vertical pixel doubling
        // for window A.
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_ADDR_WA_B0,
        // Start address for window A buffer 0.
    GFDX_ATTR_ADDR_WA_B1,
        // Start address for window A buffer 1.
    GFDX_ATTR_STRIDE_WA_B0,
        // Stride for window A buffer 0.
    GFDX_ATTR_STRIDE_WA_B1,
        // Stride for window A buffer 1.
    GFDX_ATTR_ENABLE_WA,
        // Enable/disable window A.
    GFDX_ATTR_DBL_BUF_WA,
        // Enable/disable Buf0, Buf1, VI, GE double buffer.
    GFDX_ATTR_ROTATE_BIT_WA,
        // Sets/gets GC rotate bits for window A.

    GFDX_ATTR_PALETTE_EN_WA,
        /* Enable/Disable Palette for WinA */
    GFDX_ATTR_PALETTE_WA,
        /* Set/Get Palette 666 R[31-24], G[23-16], B[15-8],
         * Index[7-0] WinA */
    GFDX_ATTR_PALETTE888_WA,
        /* Set/Get Palette 888 R[31-24], G[23-16], B[15-8],
         * Index[7-0] WinA */
    GFDX_ATTR_DV_CONTROL_EN_WA,
        /* Enable/Disable DV WinA */
    GFDX_ATTR_DV_CONTROL_WA,
        /* Set/Get DV Control R[26-24], G[18-16], B[10-8] WinA
         */


    GFDX_ATTR_COLOR_KEY0_LOWER_RANGE,
        /* Color Key0 Lower range for overlay */
    GFDX_ATTR_COLOR_KEY0_UPPER_RANGE,
        /* Color Key0 Upper range for overlay */
    GFDX_ATTR_COLOR_KEY1_LOWER_RANGE,
        /* Color Key1 Lower range for overlay */
    GFDX_ATTR_COLOR_KEY1_UPPER_RANGE,
        /* Color Key1 Upper range for overlay */
    GFDX_ATTR_COLOR_NOKEY_WA,
        /* Color Key enbled but pixel color is not in range */
    GFDX_ATTR_COLOR_NOKEY_WEIGHT0_WA,
        /* Weight0 for Color Key not match area */
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_COLOR_NOKEY_WEIGHT1_WA,
        /* Weight1 for Color Key not match area */
    GFDX_ATTR_COLOR_KEY0_1WIN_WA,
        /* Color Key0 Enble/Disable for WinA */
    GFDX_ATTR_COLOR_KEY1_1WIN_WA,
        /* Color Key1 Enble/Disable for WinA */
    GFDX_ATTR_COLOR_1WIN_WA,
        /* Blend Control for WinA where does not Overlap with
         * other Window */
    GFDX_ATTR_COLOR_1WIN_WEIGHT0_WA,
        /* Win Blend Weight0 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 0 uses
         * Weight0 */

    GFDX_ATTR_COLOR_1WIN_WEIGHT1_WA,
        /* Win Blend Weight1 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 1 uses
         * Weight1 */
    GFDX_ATTR_COLOR_KEY0_2WIN_B_WA,
        /* Color Key0 Enble/Disable for WinA, Overlap with WinB
         */
    GFDX_ATTR_COLOR_KEY1_2WIN_B_WA,
        /* Color Key1 Enble/Disable for WinA, Overlap with WinB
         */
    GFDX_ATTR_COLOR_2WIN_B_WA,
         /* Blend Control for WinA area that overlap with WinB
          * only */
    GFDX_ATTR_COLOR_2WIN_B_WEIGHT0_WA,
         /* Win Blend Weight0 for CKey disabled or CKey enabled
          * w key match area or 1-bit Alpha, Alpha 0 uses
          * Weight0 */
    GFDX_ATTR_COLOR_2WIN_B_WEIGHT1_WA,
         /* Win Blend Weight1 for CKey disabled or CKey enabled
          * w key match area or 1-bit Alpha, Alpha 1 uses
          * Weight1 */
    GFDX_ATTR_COLOR_KEY0_2WIN_C_WA,
        /* Color Key0 Enble/Disable for WinA, Overlap with WinC
         */
    GFDX_ATTR_COLOR_KEY1_2WIN_C_WA,
        /* Color Key1 Enble/Disable for WinA, Overlap with WinC
         */
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_COLOR_2WIN_C_WA,
        /* Blend Control for WinA area that overlap with WinC
         * only */
    GFDX_ATTR_COLOR_2WIN_C_WEIGHT0_WA,
        /* Win Blend Weight0 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 0 uses
         * Weight0 */
    GFDX_ATTR_COLOR_2WIN_C_WEIGHT1_WA,
        /* Win Blend Weight1 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 1 uses
         * Weight1 */
    GFDX_ATTR_COLOR_KEY0_3WIN_BC_WA,
        /* Color Key0 Enble/Disable for WinA, Overlap with WinA
         * & WinC Only */
    GFDX_ATTR_COLOR_KEY1_3WIN_BC_WA,
        /* Color Key1 Enble/Disable for WinA, Overlap with WinA
         * & WinC */
    GFDX_ATTR_COLOR_3WIN_BC_WA,
        /* Blend Control for WinA area that overlap with WinA &
         * WinC only */
    GFDX_ATTR_COLOR_3WIN_BC_WEIGHT0_WA,
        /* Win Blend Weight0 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 0 uses
         * Weight0 */
    GFDX_ATTR_COLOR_3WIN_BC_WEIGHT1_WA,
        /* Win Blend Weight1 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 1 uses
         * Weight1*/


    GFDX_ATTR_WH_WB,
        /* set/gets width height together to avoid artifacts */
    GFDX_ATTR_XY_POS_WB,
        /* set/gets x, y start position of window B */
    GFDX_ATTR_BPP_WB,
        /* current bits per pixel (8,15(RGB555),16(RGB565),
         * 18(LSB), 24(MSB)) */
    GFDX_ATTR_PIXEL_DBL_WB,
        /* On/Off Horizontal & Vertical Pixel Doubling for
         * Window B */
    GFDX_ATTR_ADDR_WB_B0,
        /* Start Address for Window B Buffer 0 */
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_ADDR_WB_B1,
        /* Start Address for Window B Buffer 1 */
    GFDX_ATTR_STRIDE_WB_B0,
        /* Stride for Window B Buffer 0 */
    GFDX_ATTR_STRIDE_WB_B1,
        /* Stride for Window B Buffer 1 */
    GFDX_ATTR_ENABLE_WB,
        /* Enable/Disable Windows B */
    GFDX_ATTR_DBL_BUF_WB,
        /* Enable/Disable Buf0, Buf1, VI, GE Double Buffer
         * (see def) */
    GFDX_ATTR_ROTATE_BIT_WB,
        /* Sets/Gets GC Rotate bits for window B */


    GFDX_ATTR_PALETTE_EN_WB,
        /* Enable/Disable Palette for WinB */
    GFDX_ATTR_PALETTE_WB,
        /* Set/Get Palette 666 R[31-24], G[23-16], B[15-8],
         * Index[7-0] WinB */
    GFDX_ATTR_PALETTE888_WB,
        /* Set/Get Palette 888 R[31-24], G[23-16], B[15-8],
        * Index[7-0] WinB */
    GFDX_ATTR_DV_CONTROL_EN_WB,
        /* Enable/Disable DV WinB */
    GFDX_ATTR_DV_CONTROL_WB,
        /* Set/Get DV Control R[26-24], G[18-16], B[10-8] WinB
         */


    GFDX_ATTR_COLOR_NOKEY_WB,
         /* Color Key enbled but pixel color is not in range */
    GFDX_ATTR_COLOR_NOKEY_WEIGHT0_WB,
         /* Weight0 for Color Key not match area */
    GFDX_ATTR_COLOR_NOKEY_WEIGHT1_WB,
         /* Weight1 for Color Key not match area */
    GFDX_ATTR_COLOR_KEY0_1WIN_WB,
         /* Color Key0 Enble/Disable for WinB */
    GFDX_ATTR_COLOR_KEY1_1WIN_WB,
         /* Color Key1 Enble/Disable for WinB */
    GFDX_ATTR_COLOR_1WIN_WB,
         /* Blend Control for WinB where does not Overlap with
          * other Window */
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_COLOR_1WIN_WEIGHT0_WB,
         /* Win Blend Weight0 for CKey disabled or CKey enabled
          * w key match area or 1-bit Alpha, Alpha 0 uses
          * Weight0 */
    GFDX_ATTR_COLOR_1WIN_WEIGHT1_WB,
         /* Win Blend Weight1 for CKey disabled or CKey enabled
          * w key match area or 1-bit Alpha, Alpha 1 uses
          * Weight1 */
    GFDX_ATTR_COLOR_KEY0_2WIN_A_WB,
        /* Color Key0 Enble/Disable for WinB, Overlap with WinA
         */
    GFDX_ATTR_COLOR_KEY1_2WIN_A_WB,
        /* Color Key1 Enble/Disable for WinB, Overlap with WinA
         */
    GFDX_ATTR_COLOR_2WIN_A_WB,
        /* Blend Control for WinB where does not Overlap with
         * other Window */
    GFDX_ATTR_COLOR_2WIN_A_WEIGHT0_WB,
        /* Win Blend Weight0 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 0 uses
         * Weight0 */
    GFDX_ATTR_COLOR_2WIN_A_WEIGHT1_WB,
        /* Win Blend Weight1 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 1 uses
         * Weight1*/


    GFDX_ATTR_COLOR_KEY0_2WIN_C_WB,
        /* Color Key0 Enble/Disable for WinB, Overlap with WinC
         */
    GFDX_ATTR_COLOR_KEY1_2WIN_C_WB,
        /* Color Key1 Enble/Disable for WinB, Overlap with WinC
         */
    GFDX_ATTR_COLOR_2WIN_C_WB,
        /* Blend Control for WinB area that overlap with WinC
         * only */
    GFDX_ATTR_COLOR_2WIN_C_WEIGHT0_WB,
        /* Win Blend Weight0 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 0 uses
         * Weight0 */
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_COLOR_2WIN_C_WEIGHT1_WB,
        /* Win Blend Weight1 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 1 uses
         *Weight1 */
    GFDX_ATTR_COLOR_KEY0_3WIN_AC_WB,
        /* Color Key0 Enble/Disable for WinB, Overlap with WinA
         * & WinC Only */
    GFDX_ATTR_COLOR_KEY1_3WIN_AC_WB,
        /* Color Key1 Enble/Disable for WinB, Overlap with WinA
         * & WinC */
    GFDX_ATTR_COLOR_3WIN_AC_WB,
         /* Blend Control for WinB area that overlap with WinA
          *& WinC only */
    GFDX_ATTR_COLOR_3WIN_AC_WEIGHT0_WB,
         /* Win Blend Weight0 for CKey disabled or CKey enabled
          * w key match area or 1-bit Alpha, Alpha 0 uses
          * Weight0 */
    GFDX_ATTR_COLOR_3WIN_AC_WEIGHT1_WB,
         /* Win Blend Weight1 for CKey disabled or CKey enabled
          * w key match area or 1-bit Alpha, Alpha 1 uses
          * Weight1*/


    GFDX_ATTR_WH_WC,
        /* set/gets width height together to avoid artifacts */
    GFDX_ATTR_XY_POS_WC,
        /* set/gets x, y start position of window C */
    GFDX_ATTR_BPP_WC,
        /* current bits per pixel (8,15(RGB555),16(RGB565),
         * 18(LSB), 24(MSB)) */
    GFDX_ATTR_TRIGGER_WENC,
       /* Set Trigger for encode window, 0- Disable, 1- Enable,
        * 2-single Shot*/
    GFDX_ATTR_PIXEL_DBL_WC,
        /* On/Off Horizontal & Vertical Pixel Doubling for
         * Window B */
    GFDX_ATTR_ADDR_WC_B0,
        /* Start Address for window C Buffer 0 */
    GFDX_ATTR_ADDR_WC_B1,
        /* Start Address for window C Buffer 1 */
    GFDX_ATTR_STRIDE_WC_B0,
        /* Stride for window C Buffer 0 */
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_STRIDE_WC_B1,
        /* Stride for window C Buffer 1 */
    GFDX_ATTR_ENABLE_WC,
        /* Enable/Disable Windows C */
    GFDX_ATTR_DBL_BUF_WC,
        /* Enable/Disable Buf0, Buf1, VI, GE Double Buffer
         * (see def) */
    GFDX_ATTR_ROTATE_BIT_WC,
        /* Sets/Gets GC Rotate bits for window C */


    GFDX_ATTR_PALETTE_EN_WC,
        /* Enable/Disable Palette for WinC */
    GFDX_ATTR_PALETTE_WC,
        /* Set/Get Palette 666 R[31-24], G[23-16], B[15-8],
         * Index[7-0] WinC */
    GFDX_ATTR_PALETTE888_WC,
        /* Set/Get Palette 888 R[31-24], G[23-16], B[15-8],
         * Index[7-0] WinC */
    GFDX_ATTR_DV_CONTROL_EN_WC,
        /* Enable/Disable DV WinC */
    GFDX_ATTR_DV_CONTROL_WC,
        /* Set/Get DV Control R[26-24], G[18-16], B[10-8] WinA
         */


    GFDX_ATTR_COLOR_NOKEY_WC,
        /* Color Key enbled but pixel color is not in range */
    GFDX_ATTR_COLOR_NOKEY_WEIGHT0_WC,
        /* Weight0 for Color Key not match area */
    GFDX_ATTR_COLOR_NOKEY_WEIGHT1_WC,
        /* Weight1 for Color Key not match area */
    GFDX_ATTR_COLOR_KEY0_1WIN_WC,
        /* Color Key0 Enble/Disable for WinC */
    GFDX_ATTR_COLOR_KEY1_1WIN_WC,
        /* Color Key1 Enble/Disable for WinC */
    GFDX_ATTR_COLOR_1WIN_WC,
        /* Blend Control for WinC where does not Overlap with
         * other Window */
    GFDX_ATTR_COLOR_1WIN_WEIGHT0_WC,
        /* Win Blend Weight0 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 0 uses
         * Weight0*/
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_COLOR_1WIN_WEIGHT1_WC,
        /* Win Blend Weight1 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 1 uses
         * Weight1 */
    GFDX_ATTR_COLOR_KEY0_2WIN_A_WC,
        /* Color Key0 Enble/Disable for WinC, Overlap with WinA
         */
    GFDX_ATTR_COLOR_KEY1_2WIN_A_WC,
        /* Color Key1 Enble/Disable for WinC, Overlap with WinA
         */
    GFDX_ATTR_COLOR_2WIN_A_WC,
        /* Blend Control for WinC area that overlap with WinA
         * only */
    GFDX_ATTR_COLOR_2WIN_A_WEIGHT0_WC,
        /* Win Blend Weight0 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 0 uses
         * Weight0 */
    GFDX_ATTR_COLOR_2WIN_A_WEIGHT1_WC,
        /* Win Blend Weight1 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 1 uses
         * Weight1*/
    GFDX_ATTR_COLOR_KEY0_2WIN_B_WC,
        /* Color Key0 Enble/Disable for WinC, Overlap with WinB
         */
    GFDX_ATTR_COLOR_KEY1_2WIN_B_WC,
        /* Color Key1 Enble/Disable for WinC, Overlap with WinB
         */
    GFDX_ATTR_COLOR_2WIN_B_WC,
         /* Blend Control for WinC area that overlap with WinB
          *only */
    GFDX_ATTR_COLOR_2WIN_B_WEIGHT0_WC,
         /* Win Blend Weight0 for CKey disabled or CKey enabled
          * w key match  area or 1-bit Alpha, Alpha 0 uses
          * Weight0 */
    GFDX_ATTR_COLOR_2WIN_B_WEIGHT1_WC,
         /* Win Blend Weight1 for CKey disabled or CKey enabled
          * w key match area or 1-bit Alpha, Alpha 1 uses
          * Weight1 */
    GFDX_ATTR_COLOR_KEY0_3WIN_AB_WC,
         /* Color Key0 Enble/Disable for WinC, Overlap with WinA
          * & WinB Only */
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_COLOR_KEY1_3WIN_AB_WC,
        /* Color Key1 Enble/Disable for WinC, Overlap with WinA
         * & WinB */
    GFDX_ATTR_COLOR_3WIN_AB_WC,
        /* Blend Control for WinC area that overlap with WinA
         *& WinB only */
    GFDX_ATTR_COLOR_3WIN_AB_WEIGHT0_WC,
        /* Win Blend Weight0 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 0 uses
         *Weight0 */
    GFDX_ATTR_COLOR_3WIN_AB_WEIGHT1_WC,
        /* Win Blend Weight1 for CKey disabled or CKey enabled
         * w key match area or 1-bit Alpha, Alpha 1 uses
         * Weight1*/


    GFDX_ATTR_COLOR_KEY,
         /* GC Color Key for overlay */
    GFDX_ATTR_COLOR_KEY_MASK,
         /* GC Color Key Mask for overlay */
    GFDX_ATTR_OVERLAP_TYPE,
         /* Set overlap type (see def) */
    GFDX_ATTR_V_DISPLAY_END,
         /* Gets VDisplay End for GE Start Window programming
          */


    GFDX_ATTR_PWM0_CLK_RANGE,
         /*RO Gets range for clock divider L=0:H=15 */
    GFDX_ATTR_PWM0_DTY_RANGE,
         /*RO Gets range for duty cycle L=0:H=255 */
    GFDX_ATTR_PWM0_CLOCK,
        /* Controls clock divider for PWM0 (FP0F[7:4]) */
    GFDX_ATTR_PWM0_DUTYCY,
        /* Controls duty cycle for PWM0 (FP0F[15:8]) */
    GFDX_ATTR_PWM1_CLK_RANGE,
        /*RO Gets range for clock divider L=0:H=15 */
    GFDX_ATTR_PWM1_DTY_RANGE,
        /*RO Gets range for duty cycle L=0:H=255 */
    GFDX_ATTR_PWM1_CLOCK,
        /* Controls clock divider for PWM1 (FP0F[23:20]) */
    GFDX_ATTR_PWM1_DUTYCY,
        /* Controls duty cycle for PWM1 (FP0F[31:24]) */
```

GFDXATTRIBUTES Enumeration Type (continued)

```
    GFDX_ATTR_DISP_TYPE,
        /* LCD Type 0-Main Lcd, 1-Sub Lcd */
    GFDX_ATTR_SW_REG_1,
        /* SW register 1 for program usage */
    GFDX_ATTR_SW_REG_2,
        /* SW register 2 for program usage */
    GFDX_ATTR_WH_WENC,
        /* set/gets width height of screen encode window */
    GFDX_ATTR_XY_POS_WENC,
        /* set/gets x, y start position of screen encode window
         */
    GFDX_ATTR_ENABLE_WENC,
        /* Enable/Disable screen encode window */
    GFDX_ATTR_DEVICE_INFO,
        /* Get Device ID & Rev */
    GFDX_ATTR_CRC_ENABLE,
        /* Enable CRC, Set Wait 1 or 2 VSync */
    GFDX_ATTR_CRC_CHECK_SUM,
        /* Get Check Sum */
/* New for Hardware Cursor  */
    GFDX_ATTR_HC_SHAPE,
        /* load Hardware Cursor Image */
    GFDX_ATTR_HC_HOTSPOT,
        /* load Hardware Cursor Image */
    GFDX_ATTR_HC_COLOR,
        /* set foreground and background color */
    GFDX_ATTR_HC_POSITION,
        /* Set Hardware Cursor new Position */
    GFDX_ATTR_HC_SHOW,
        /* Enable, Disable Hardware Cursor */
    GFDX_ATTR_HC_RELEASE
        /* free internal buffer */

} GFDXATTRIBUTES;
```

Initial Programming Sequence

The best way to learn and use the GFDxAPI is to refer to the sample application source code that comes with the GFSDK package. True beginners might start a new application by first creating one of their own at the same directory level as the Demo (or Samples) application source directory. This way the same directory tree structure can be maintained. The general sequence for programming the GFDxAPI follows.

1. Set the main panel type (and the subpanel type if subpanel support is enabled) and other configuration flags in GFPlat.h or GFExtCfg.h, and compile the required project.
2. Display application should call **`GFRmOpen()`** to get a resource manager handle to resource services like memory surface allocation, and register and memory access.
3. The application should call the **`GFRmComponentGet()`** function to get a **`DxHandle`** to access the GFDxAPI and to initialize the GFDxAPI function call pointers.
4. If multiple panel support is enabled, use **`GFDxSetAttribute()`** to set the main panel type (and the subpanel type if subpanel support is enabled). This function should be called twice if a subpanel is used.
5. Calling the **`GFDxGetProperty()`** function returns device information and main LCD and sub-LCD size information.
6. The application should call **`GFDxInit()`** to initialize clock sources, clocks (PLL), the oscillator for the PLL and relaxation oscillator, thresholds, and SRAM parameters.
7. The application should call **`GFDxSetDisplay()`** with the **`LCD_MAIN`** parameter to initialize the display controller, the main panel timings, and the GPIO.

   The application can call **`GFDxSetDisplay()`** with the **`LCD_SUB`** parameter to initialize the subpanel timings and GPIO.
8. **`GFRmSurfaceAlloc()`** can be called to allocate the primary surface for the main panel (and subpanel if enabled). Additionally, **`GFDxSetAttributes()`** can be called to modify attributes related to the LCD and primary surfaces, such as the start address, stride, and **`bpp`**. Also use **`GFDxSwitchLCD()`** to set the main display type or the subpanel (if subpanel support is enabled). This function should be called if both main and subpanels are to be used.

After allocating both surfaces, the application can access any surface any time

9. The display is now up and the frame buffer can be accessed.

The following steps are required only before rebooting a system.

10. **`GFRmSurfaceFree()`** frees the main and subpanel surface memory allocated in the display buffer. This function should be called twice if a surface was also allocated for a subpanel display.
11. **`GFRmComponentRelease()`** frees the display component handle, **`DxHandle`**.
12. **`GFRmClose()`** closes the resource manager. All necessary cleaning up is performed here. The application is required to call this function.

CHAPTER

7

Graphics API (GFGxAPI)

Overview

The graphics API (GFGxAPI) is designed for NVIDIA media processors, which have hardware 2D accelerated graphics engines that support drawing operations such as these:

- 2D bit-block-transfer (bitblit or simply blit) operations
  - Rectangle fill
  - Screen-to-screen blit
  - Monochrome-to-color expansion blit
  - Hardware-assist system-memory-to-display-memory transfer blit
  - 8x8 monochrome (1-bpp) pattern blit
  - Transparency support
- Bresenham line drawing operation
- 256 ternary raster operation codes (see "Raster Operation (ROP) Codes" on page 214)

The GFGxAPI is a stateless API[1]: It does not maintain any context information.

1. This policy has been in effect since the Beta 3 release on Feb. 28, 2005.

Unless noted otherwise, all GFGxAPI operations work on 8-bpp (indexed) and 16-bpp (RGB565) destination color depths. The GFGxAPI always assumes a two-dimensional Cartesian coordinate system. The video memory is always organized with the (*x*, *y*) coordinate (0, 0) at the top-left corner of memory (the memory starting location is at address offset 0). The figure below shows the coordinate system with width **`w`** and height **`h`**.

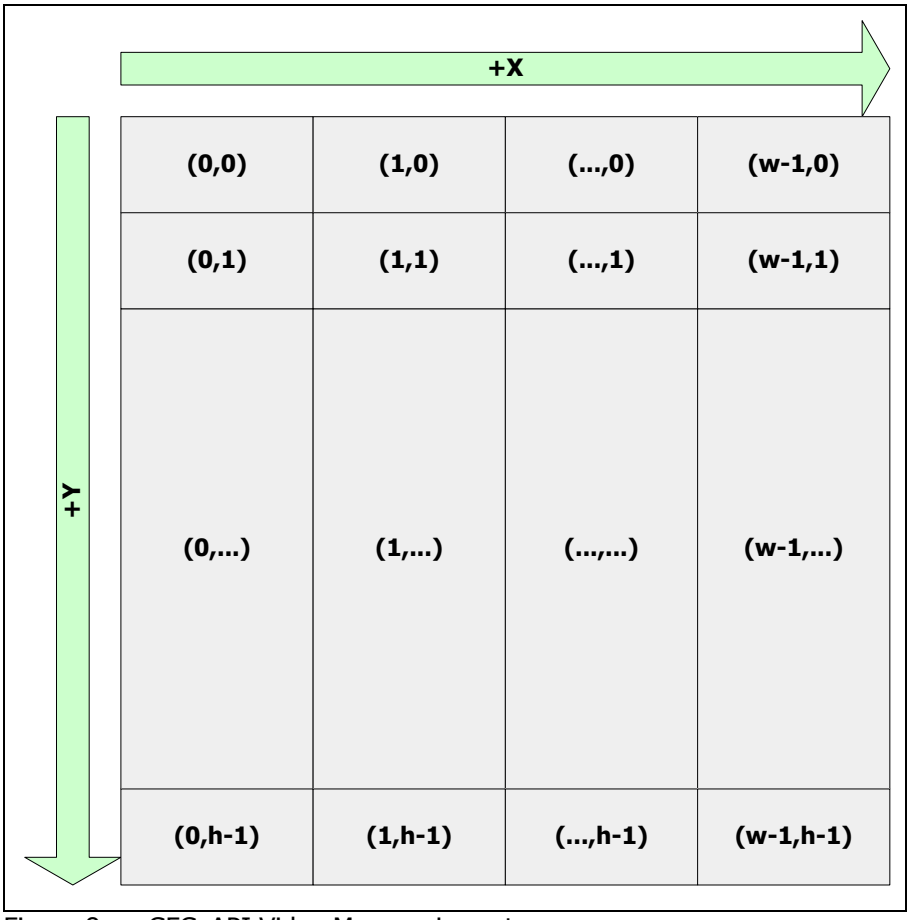


Figure 8. GFGxAPI Video Memory Layout

GFGxAPI Reference

The graphics API is detailed in the following sections:

GFGxAPI Functions

Most of the GFGxAPI is a thin layer of routines that directly access the graphics engine of the NVIDIA media processor. This direct access provides quick acceleration without complex register-level programming. The graphics engine supports two primitive rendering capabilities: blit (Bit-Block-Transfer) and line drawing.

The GFGxAPI functions are as follows:

GFGxGetProperty()

This function returns additional information about the display hardware in the **GFPROPERTY** structure (see "GFPROPERTY" on page 10). It is recommended, but not mandatory, that you call this function to query for the GFGxAPI version and capabilities before proceeding to call any of the other GFGxAPI functions (see "GFPROPERTY Definitions" below).

The GoForce 2D engine supports the following capabilities: blitting, line drawing, clipping (outside a rectangle), stretch blit, alpha blending (GoForce 4000 Rev B and GoForce 3D 4800), and alpha blending with color

transparency (GoForce 3D 4800). In the GoForce 4000 Rev B, the alpha blending and color transparency operations are mutually exclusive.

Function Prototype

```
GF_RETTYPE  GFGxGetProperty(
    GF_HANDLE    GxHandle,
    PGFPROPERTY  pGxProp );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
pGxProp	Pointer to a GFPROPERTY structure.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

The GFGxAPI **GFPROPERTY** capabilities are defined below.

GFPROPERTY Definitions

```
/*
    GX Capability Definitions
*/
#define GFGX_CAP_BLT                          0x00000001UL
    // Rectangle fill, copy rectangle, and blit available.
#define GFGX_CAP_LINE                         0x00000002UL
    // Line blitting available.
#define GFGX_CAP_CLIP                         0x00000004UL
    // Clipping available.
#define GFGX_CAP_STRETCHBLT                   0x00000008UL
    // Indicates GFGxStretchBlt() is available.
#define GFGX_CAP_ALPHABLENDING                0x00000010UL
    // Alpha blending available.
#define GFGX_CAP_ALPHA_COLOR_CMP              0x00000020UL
    // Alpha blending and transparency available.
#define GFGX_CAP_FAST_ROTATION                0x00000040UL
    // Fast Rotation
#define GFGX_CAP_CLIPPING_INSIDE              0x00000100UL
    // Inside Clipping
#define GFGX_CAP_FADE_BLT                     0x00000200UL
    // Fade Blt
```

GFPROPERTY Definitions (continued)

```
#define GFGX_CAP_MASK_BLT                       0x00000800UL
    // Mask Blt
#define GFGX_CAP_COLOR_PATTERN                  0x00001000UL
    // Color Pattern
#define GFGX_CAP_CLR_PAT_TRANSPARENT            0x00002000UL
    // Color Pattern Transparency
#define GFGX_CAP_ALPHA_AND_FADING               0x00004000UL
    // ALpha Blending & fading
#define GFGX_CAP_ALPHA_PLANAR_1BPP              0x00008000UL
    // Planar 1  BPP Alpha
#define GFGX_CAP_ALPHA_PLANAR_2BPP              0x00010000UL
    // Planar 2  BPP Alpha
#define GFGX_CAP_ALPHA_PLANAR_4BPP              0x00020000UL
    // Planar 4  BPP Alpha
#define GFGX_CAP_ALPHA_PLANAR_8BPP              0x00040000UL
    // Planar 8  BPP Alpha
#define GFGX_CAP_ALPHA_PLANAR_SRC4DST4          0x00080000UL
    // Planar 44 BPP Alpha
    // 8 bpp plane, src * 4 + dst*4
#define GFGX_CAP_ALPHA_PLANAR_32BLEND16         0x00200000UL
    // 32 bits source blending
#define GFGX_CAP_ALPHA_SRC1555                  0x00400000UL
    // Src alpha 1555
#define GFGX_CAP_ALPHA_SRC4444                  0x00800000UL
    // Src alpha 4444
#define GFGX_CAP_ALPHA_FIXED                    0x01000000UL
    // Fixed alpha
```

GFGxFillRect()

This function fills a rectangle in video memory with the specified solid color.

Function Prototype

```
GF_RETTYPE  GFGxFillRect(
    GF_HANDLE  GxHandle,
    NvS16      x,
    NvS16      y,
    NvS16      w,
    NvS16      h,
    NvU32      color );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
x	Destination *x* in pixels (relative to left).
y	Destination y in scan lines (relative to top).
w	Width of the rectangle in pixels.
h	Height of the rectangle in scan lines.
color	Solid color (index or RGB565).

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyRect()

This function is commonly used in scrolling an image in video memory. It copies a rectangular image in the video memory to another location in the same video memory.

GFGxCopyRect() checks for destruction of the copied image in video memory caused by overlapping source and destination locations. Also, if the destination for the rectangle exceeds the limit of the video memory, wrapping is done in hardware and no error code is returned.

Function Prototype

```
GF_RETTYPE  GFGxCopyRect(
    GF_HANDLE  GxHandle,
    NvS16      dx,
    NvS16      dy,
    NvS16      w,
    NvS16      h,
    NvS16      sx,
    NvS16      sy );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
dx	Destination *x* in pixels (relative to left).
dy	Destination *y* in scan lines (relative to top).
w	Width of the rectangle in pixels.

Parameters

h	Height of the rectangle in scan lines.
sx	Source *x* in pixels (relative to left).
sy	Source *y* in scan lines (relative to top).

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyRectDirect()

This function is a special case of **GFGxCopyRect()**. It does not check for any destruction of the copied image in video memory caused by overlapping source and destination locations.

Function Prototype

```
GF_RETTYPE  GFGxCopyRectDirect(
    GF_HANDLE  GxHandle,
    NvS16      dx,
    NvS16      dy,
    NvS16      w,
    NvS16      h,
    NvS16      sx,
    NvS16      sy );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
dx	Destination *x* in pixels (relative to left).
dy	Destination *y* in scan lines (relative to top).
w	Width of the rectangle in pixels.
h	Height of the rectangle in scan lines.
sx	Source *x* in pixels (relative to left).
sy	Source *y* in scan lines (relative to top).

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyMonoBitmap()

The source bitmap is specified in monochrome (1-bpp) pixels. The source monochrome bitmap resides in system memory.

Function Prototype

```
GF_RETTYPE  GFGxCopyMonoBitmap(
    GF_HANDLE  GxHandle,
    NvS16      dx,
    NvS16      dy,
    NvS16      w,
    NvS16      h,
    NvS16      sx,
    NvS16      sy,
    NvU16      fgColor,
    NvU16      bgColor,
    NvS16      srcStride,
    NvU8       *pMonoBits );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
dx	Destination *x* in pixels (relative to left).
dy	Destination *y* in scan lines (relative to top).
w	Width of the rectangle in pixels.
h	Height of the rectangle in scan lines.
sx	Source *x* in pixels (relative to left of source bitmap).
sy	Source *y* in scan lines (relative to top of source bitmap).
fgColor	Color applied to source data bit 1. The color depth must be same as the frame buffer.
bgColor	Color applied to source data bit 0. The color depth must be same as the frame buffer.
srcStride	Source stride in bytes.
*pMonoBits	Pointer to monochrome (1-bpp) bitmap.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyTransMonoBitmap()

This function is similar to **`GFGxCopyMonoBitmap()`** except for the **`selectTrans`** parameter. This parameter indicates whether bit data equal to 0 or 1 is going to be transparent. This function could be used for drawing a transparent text string or character.

Function Prototype

```
GF_RETTYPE  GFGxCopyTransMonoBitmap(
    GF_HANDLE  GxHandle,
    NvS16      dx,
    NvS16      dy,
    NvS16      w,
    NvS16      h,
    NvS16      sx,
    NvS16      sy,
    NvU16      color,
    NvS16      srcStride,
    NvU8       *pMonoBits,
    NvS16      selectTrans );
```

Parameters

`GxHandle`	Handle specific to the GFGxAPI.
`dx`	Destination *x* in pixels (relative to left).
`dy`	Destination *y* in scan lines (relative to top).
`w`	Width of the rectangle in pixels.
`h`	Height of the rectangle in scan lines.
`sx`	Source *x* in pixels (relative to left of source bitmap).
`sy`	Source *y* in scan lines (relative to top of source bitmap).
`color`	Color applied to non-transparent source data bit.
`srcStride`	Source stride in bytes.
`*pMonoBits`	Pointer to monochrome (1-bpp) bitmap.
`selectTrans`	Transparency select: 0 or 1.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFGxCopyColorBitmap()

This function transfers a color bitmap from system memory to video memory. The source bitmap's color depth *must* match the destination video memory's color depth.

Function Prototype

```
GF_RETTYPE  GFGxCopyColorBitmap(
    GF_HANDLE  GxHandle,
    NvS16      dx,
    NvS16      dy,
    NvS16      w,
    NvS16      h,
    NvS16      sx,
    NvS16      sy,
    NvS16      srcStride,
    NvU8       *pColorBits );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
dx	Destination *x* in pixels (relative to left).
dy	Destination *y* in scan lines (relative to top).
w	Width of the rectangle in pixels.
h	Height of the rectangle in scan lines.
sx	Source *x* in pixels (relative to left of source bitmap).
sy	Source *y* in scan lines (relative to top of source bitmap).
srcStride	Source stride in bytes.
*pColorBits	Pointer to color bitmap that resides in system memory.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyPackedColorBitmap()

This is a special case of **GFGxCopyColorBitmap()**. This function transfers a bitmap from system memory to video memory with certain restrictions:

- The source bitmap pointer, **pColorBits**, has to be a 32-bit (DWORD) aligned address.
- The size of each line of the source bitmap must be 64-bit (8 bytes, QWORD) aligned.
- No bytes may be skipped between lines.
- The source bitmap's color depth *must* match the destination video memory's color depth.

Function Prototype

```
GF_RETTYPE  GFGxCopyPackedColorBitmap(
    GF_HANDLE  GxHandle,
    NvU32      *pColorBits,
    NvU16      bitmapSize,
    GFRECT     rect );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
*pColorBits	Pointer to color bitmap.
bitmapSize	Total bytes to transfer.
rect	Destination rectangle. See "GFRECT" on page 11.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxBltSurface()

This function performs a surface-to-surface blit. The color depth of the source surface must match that of the destination. Also, the width and height of the source must be the same as the destination's width and height. Clipping is *not* supported by this function.

Function Prototype

```
GF_RETTYPE  GFGxBltSurface((
    GF_HANDLE     GxHandle,
    PGFRMSURFACE  *ppDestSurf,
    PGFRMSURFACE  *ppSrcSurf,
    NvS16         w,
    NvS16         h,
```

Function Prototype (continued)

```
    NvU16          rop,
    NvU32          srcStartAddr,
    NvU32          dstStartAddr,
    NvU32          flag,
    NvU32          colorCompare );
```

Parameters

`GxHandle`	Handle specific to the GFGxAPI.
`*ppDestSurf`	Pointer to destination surface pointer.
`*ppSrcSurf`	Pointer to source surface pointer.
`w`	Width of the rectangle in pixels.
`h`	Height of the rectangle in scan lines.
`rop`	Raster operation code. See "Raster Operation (ROP) Codes" on page 214 for more information.
`srcStartAddr`	Start address relative to the source surface.
`dstStartAddr`	Start address relative to the destination surface.
`flag`	Transparent blit flag, which is one of the following: • `GFGX_BLT_TRANSPARENT_SRC_COLOR` • `GFGX_BLT_TRANSPARENT_SRC_MONO` • `GFGX_BLT_TRANSPARENT_SRC_MONO_INV` • `GFGX_BLT_TRANSPARENT_DST` See "Transparent Flags" in "GFGXBLTPARAM Definitions" on page 206 for definitions of these flags. If there is no transparency, `flag` is 0.
`colorCompare`	Color key for transparent blit. If there is no transparent blit, `colorCompare` is 0.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFGxBlt()

This function performs a complicated blit function that is the super set of the **`GFGxCopyRect()`**, **`GFGxFillRect()`**, **`GFGxCopyMonoBitmap()`**, **`GFGxCopyTransMonoBitmap()`**, **`GFGxCopyColorBitmap()`**, and

GFGxBltSurface() functions. See the description below for more information.

Function Prototype

```
GF_RETTYPE  GFGxBlt(
    GF_HANDLE     GxHandle,
    GFGXBLTPARAM  *pBltParam );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
*pBltParam	See "GFGxAPI Data Structure: GFGXBLTPARAM" on page 203.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Description. **GFGxBlt()** supports both raster operations (see "Common Raster Operations" on page 168) and alpha blending operations ("Common Alpha Blending Operations" on page 171). The two types of operations are mutually exclusive.

The following three fields in the **GFGXBLTPARAM** structure have to be specified every time for both raster and alpha blending operations.

- **w**
- **h**
- **flags**

Definition bits for **flags** can be grouped as follow:

- **GFGX_BLT_TRANSPARENT_*****

  These flags belongs to transparency control. *One and only one* of these flags must be set when some form of transparency is required.
- **GFGX_BLT_SRC_*****

  When a source operand is required, *one and only one* of these flags has to be set.
- **GFGX_BLT_PAT_*****

  When a pattern operand is required, *one and only one* of these flags has to be set.
- **GFGX_BLT_CLIP**

Applies clipping on the blit operations. The rectangle specified by **GFGxSetClip()** is the bounding clipping rectangle.

Clipping is *not* effective if either of the blit flags **GFGX_BLT_SURFACE** or **GFGX_BLT_MODE_LINEAR** is enabled. It works only in the XY-mode blit, which is the default mode.

- **GFGX_BLT_SURFACE**

  Enables surface-to-surface blit operations.

- **GFGX_BLT_MODE_*****

  These flags specify whether the blit is linear or XY. *One and only one* of these flags has to be specified.

- **GFGX_BLT_SET_DST_STRIDE**

  This flag has to be specified if the destination stride must be changed.

- **GFGX_LINE_NOT_DRAW_LAST_PIXEL**

  This flag specifies that the last pixel of a line is not to be drawn.

- **GFGX_BLT_ALPHA_*****

  These flags control alpha blending. The **GFGX_BLT_ALPHA_BLENDING** flag always has to be specified with the other alpha blending flags to select different alpha blending formats.

  The GoForce 3D 4800 supports the capability of simultaneous alpha blending and color transparency. Earlier GoForce media processors do not support this feature.

Setting **flags** specifies different types of blits. If the source **stride** is negative, the source bitmap is upside down and ***pBits** must point to the bottom part of the rectangle. For blit operations involving the transfer of a monochrome (1-bpp) source bitmap, color expansion is performed using **fgColor** and **bgColor** for monochrome pixel values of 1 and 0 respectively. Similarly, monochrome (1-bpp) 8x8 pattern data is expanded into **patFgColor** and **patBgColor**.

In addition to the **w**, **h**, and **flags** fields, the **rop3** field in the **GFGXBLTPARAM** structure must be specified every time for raster operations. The field specifies a ternary raster operation (ROP3) code. Further information on ROP3 codes is found in "Raster Operation (ROP) Codes" on page 214.

Common Raster Operations

This section contains common raster operations that can be performed by specifying and combining **`flags`** bits and other fields in **`GFGXBLTPARAM`**. *Fields* are additional settings that need to be filled in the **`GFGXBLTPARAM`** structure for a particular operation. *ROP3* contains valid ROP3 operations.

- **Fill a rectangular destination area in video memory using a solid pattern.** This is a superset of **`GFGxFillRect()`**.

  Flag. `GFGX_BLT_PAT_SOLID`

  Fields. `dx, dy, patFgColor`

  ROP3. `0xF0` is common. ROP3s involving pattern and destination operands.

- **Transfer a monochrome (1-bpp) source bitmap in system memory to video memory.** This is a superset of **`GFGxCopyMonoBitmap()`**.

  Flag. `GFGX_BLT_SRC_SYSMEM_MONO`

  Fields. `sx, sy, dx, dy, *pBits, fgColor, bgColor, stride, bpp`

  ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Transfer a monochrome (1-bpp) source bitmap to video memory with transparency control.** All source bits of 0 are transparent.

  Flag. `GFGX_BLT_SRC_SYSMEM_MONO, GFGX_BLT_TRANSPARENT_MONO`

  Fields. `sx, sy, dx, dy, *pBits, stride, bpp`

  ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Transfer a color source bitmap to video memory.** The source bitmap color depth must match the destination color depth in video memory.
  - **XY mode.**

  Flag. `GFGX_BLT_SRC_SYSMEM_COLOR`

  Fields. `sx, sy, dx, dy, *pBits, stride, bpp`

  ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

  - **XY mode if destination stride must be changed.**

  Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_SET_DST_STRIDE`

  Fields. `sx, sy, dx, dy, *pBits, stride, dstStride, bpp`

  ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Linear mode.**

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_MODE_LINEAR`

Fields. `srcStartAddr, dstStartAddr, stride, *pBits, bpp`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Linear mode if destination stride must be changed.**

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_MODE_LINEAR, GFGX_SET_DST_STRIDE`

Fields. `srcStartAddr, dstStartAddr, stride, dstStride, *pBits, bpp`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Transfer a color source bitmap to video memory by comparing a source pixel against the color key.** (The color key is parameter **colorCompare**). If they match, the destination pixel location is *not* overwritten. The source bitmap, the destination video memory, and the color key must have the same color depth.
  - **XY mode.**

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_TRANSPARENT_SRC_COLOR`

Fields. `sx, sy, dx, dy, *pBits, stride, bpp, colorCompare`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

  - **XY mode if destination stride must be changed.**

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_SET_DST_STRIDE, GFGX_BLT_TRANSPARENT_SRC_COLOR`

Fields. `sx, sy, dx, dy, *pBits, stride, dstStride, bpp, colorCompare`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

  - **Linear mode.**

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_MODE_LINEAR, GFGX_BLT_TRANSPARENT_SRC_COLOR`

Fields. `srcStartAddr, dstStartAddr, stride, *pBits, bpp, colorCompare`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Linear mode if destination stride must be changed.**

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_MODE_LINEAR, GFGX_SET_DST_STRIDE, GFGX_BLT_TRANSPARENT_SRC_COLOR`

Fields. `srcStartAddr, dstStartAddr, stride, dstStride, *pBits, bpp, colorCompare`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Blit an 8x8 monochrome (1-bpp) pattern.**

Flag. `GFGX_BLT_PAT_MONO`

Fields. `dx, dy, px, py, pattern0, pattern1, patFgColor, patBgColor`

ROP3. `0xF0` is common. Some ROP3s with pattern and destination operands.

- **Transfer a color source bitmap to video memory with ROP3 codes involving a monochrome (1-bpp) 8x8 pattern.** This is a complex 3-operand operation.

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_PAT_MONO`

Fields. `sx, sy, dx, dy, *pBits, stride, bpp, px, py, pattern0, pattern1, patFgColor, patBgColor`

ROP3. ROP3s involving source, pattern and destination operands.

- **Perform surface-to-surface blit operations.** The source and destination surfaces can be in either video or system memory. If both the source and destination surfaces are in system memory, only ROP code `0xCC` is supported. The color depth of the source surface should be equal to that of the destination surface.

Flag. `GFGX_BLT_SURFACE`

Fields. `*ppDestSurf, *ppSrcSurf, srcStartAddr, dstStartAddr, sw, sh`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Perform video-memory-to-video-memory blit operations.**
  - **XY mode.**

Flag. `GFGX_BLT_SRC_VIDEO`

Fields. `sx, sy, dx, dy`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **XY mode if destination stride must be changed.**

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_SET_DST_STRIDE`

Fields. `sx, sy, dx, dy, dstStride`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Linear mode.**

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_MODE_LINEAR`

Fields. `srcStartAddr, dstStartAddr`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

- **Linear mode if destination stride must be changed.**

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_MODE_LINEAR, GFGX_SET_DST_STRIDE`

Fields. `srcStartAddr, dstStartAddr, dstStride`

ROP3. `0xCC` is common. Some ROP3s with source and destination operands.

Common Alpha Blending Operations

The following alpha blending modes are supported:

- **Fixed/inverse fixed alpha**. The blit blends a source with the destination using a fixed or inverse alpha factor.
- **Source 1555 alpha/inverse source 1555 alpha**. The blit blends a source with the destination using a fixed foreground alpha and a fixed background alpha factor that are selected by `Src[15]`.
- **Source 4444/inverse source 4444 alpha**. The blit blends a source with the destination using a four-bit alpha factor that is extracted from `Src[15:12]` for each pixel.

This section contains common alpha blending operations that can be performed by specifying and combining **`flags`** bits and other fields in **`GFGXBLTPARAM`**. *Fields* are additional settings that need to be filled in the **`GFGXBLTPARAM`** structure for a particular operation. ROP3 operations cannot be used.

- **Transfer a monochrome (1-bpp) source bitmap in system memory to video memory.**

Flag. `GFGX_BLT_SRC_SYSMEM_MONO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/GFGX_BLT_ALPHA_FIXED_INV`

Fields. `sx, sy, dx, dy, *pBits, fgColor, bgColor, stride, bpp, alpha`

Flag. `GFGX_BLT_SRC_SYSMEM_MONO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV`

Fields. `sx, sy, dx, dy, *pBits, fgColor, bgColor, stride, bpp, alpha, alphaBg`

Flag. `GFGX_BLT_SRC_SYSMEM_MONO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV`

Fields. `sx, sy, dx, dy, *pBits, fgColor, bgColor, stride, bpp`

- **Transfer a source bitmap to video memory**. The source bitmap color depth must match the destination color depth in video memory.
  - **XY Mode**.

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/GFGX_BLT_ALPHA_FIXED_INV`

Fields. `sx, sy, dx, dy, *pBits, stride, bpp, alpha`

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV`

Fields. `sx, sy, dx, dy, *pBits, stride, bpp, alpha, alphaBg`

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV`

Fields. `sx, sy, dx, dy, *pBits, stride, bpp`

  - **XY mode if destination stride must be changed**.

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/GFGX_BLT_ALPHA_FIXED_INV, GFGX_BLT_SET_DST_STRIDE`

Fields. `sx, sy, dx, dy, *pBits, stride, bpp, alpha`

Flag. `GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV, GFGX_BLT_SET_DST_STRIDE`

Fields. sx, sy, dx, dy, *pBits, stride, bpp, alpha, alphaBg, dstStride

Flag. GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV, GFGX_BLT_SET_DST_STRIDE

Fields. sx, sy, dx, dy, *pBits, stride, bpp, dstStride

- **Linear mode**.

Flag. GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_MODE_LINEAR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/ GFGX_BLT_ALPHA_FIXED_INV

Fields. srcStartAddr, dstStartAddr, stride, *pBits, bpp

Flag. GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV

Fields. srcStartAddr, dstStartAddr, stride, *pBits, bpp, alpha, alphaBg

Flag. GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV

Fields. srcStartAddr, dstStartAddr, stride, *pBits, bpp

- **Linear mode if destination stride must be changed**.

Flag. GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_MODE_LINEAR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/ GFGX_BLT_ALPHA_FIXED_INV, GFGX_SET_DST_STRIDE

Fields. srcStartAddr, dstStartAddr, stride, *pBits, bpp, dstStride, alpha

Flag. GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV, GFGX_SET_DST_STRIDE

Fields. srcStartAddr, dstStartAddr, stride, *pBits, bpp, dstStride, alpha, alphaBg

Flag. GFGX_BLT_SRC_SYSMEM_COLOR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV

Fields. `srcStartAddr, dstStartAddr, stride, *pBits, bpp, dstStride`

- **Perform surface-to-surface blit operations.** The source and destination surfaces can be in either video or system memory. The color depth of the source surface should be equal to that of the destination surface.

Flag. `GFGX_BLT_SURFACE, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/GFGX_BLT_ALPHA_FIXED_INV`

Fields. `*ppDestSurf, *ppSrcSurf, srcStartAddr, dstStartAddr, sw, sh, alpha`

Flag. `GFGX_BLT_SURFACE, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV`

Fields. `*ppDestSurf, *ppSrcSurf, srcStartAddr, dstStartAddr, sw, sh, alpha, alphaBg`

Flag. `GFGX_BLT_SURFACE,GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV`

Fields. `*ppDestSurf, *ppSrcSurf, srcStartAddr, dstStartAddr, sw, sh`

- **Perform video-memory-to-video-memory blit operations.**
  - **XY Mode**.

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/GFGX_BLT_ALPHA_FIXED_INV`

Fields. `sx, sy, dx, dy ,alpha`

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV`

Fields. `sx, sy, dx, dy ,alpha, alphaBg`

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV`

Fields. `sx, sy, dx, dy`

- **XY mode if destination stride must be changed**.

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/GFGX_BLT_ALPHA_FIXED_INV, GFGX_SET_DST_STRIDE`

Fields. `sx, sy, dx, dy ,alpha, dstStride`

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV, GFGX_SET_DST_STRIDE`

Fields. `sx, sy, dx, dy, alpha, alphaBg, dstStride`

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV, GFGX_SET_DST_STRIDE`

Fields. `sx, sy, dx, dy, dstStride`

- **Linear mode**.

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_MODE_LINEAR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/ GFGX_BLT_ALPHA_FIXED_INV`

Fields. `srcStartAddr, dstStartAddr, alpha`

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV`

Fields. `srcStartAddr, dstStartAddr, alpha, alphaBg`

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/GFGX_BLT_ALPHA_SRC_4444_INV`

Fields. `srcStartAddr, dstStartAddr`

- **Linear mode if destination stride must be changed**.

Flag. `GFGX_BLT_SRC_VIDEO, GFGX_BLT_MODE_LINEAR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_FIXED/ GFGX_BLT_ALPHA_FIXED_INV, GFGX_SET_DST_STRIDE`

Fields. `srcStartAddr, dstStartAddr, dstStride, alpha`

Flag. GFGX_BLT_SRC_VIDEO, GFGX_BLT_MODE_LINEAR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_1555_T/ GFGX_BLT_ALPHA_SRC_1555_T_INV, GFGX_SET_DST_STRIDE

Fields. srcStartAddr, dstStartAddr, dstStride, alpha, alphaBg

Flag. GFGX_BLT_SRC_VIDEO, GFGX_BLT_MODE_LINEAR, GFGX_BLT_ALPHA_BLENDING, GFGX_BLT_ALPHA_SRC_4444/ GFGX_BLT_ALPHA_SRC_4444_INV, GFGX_SET_DST_STRIDE

Fields. srcStartAddr, dstStartAddr, dstStride

GFGxLine()

This function draws a line from a point (*x1, y1*) to another point (*x2, y2*) with a specified color and ROP2 code. (See "Raster Operation (ROP) Codes" on page 214 for all ROP2 codes.) **GFGxSetClip()** should be called before calling **GFGxLine()**. If clipping is needed, it is assumed that clipping is enabled and clipping rectangle was set by **GFGxSetClip()**. If clipping is not set, the line wraps if it is too long, and no error code is returned.

Function Prototype

```
GF_RETTYPE  GFGxLine(
    GF_HANDLE  GxHandle,
    NvU16      x1,
    NvU16      y1,
    NvU16      x2,
    NvU16      y2,
    NvU16      rop2,
    NvU32      color,
    NvU16      flags );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
x1	Destination x1 in pixels (relative to left).
y1	Destination y1 in scan lines (relative to top).
x2	Destination x2 in pixels (relative to left).
y2	Destination y2 in scan lines (relative to top).
rop2	ROP2 code.

Parameters (continued)

color	Color key for transparent blit or pattern color.
flags	Set to GFGX_LINE_NOT_DRAW_LAST_PIXEL if the last pixel is not to be drawn.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Line Draw Flag

```
#define GFGX_LINE_NOT_DRAW_LAST_PIXEL       0x00000001UL
```

GFGxSetClip()

This function enables or disables clipping and also sets the clipping rectangle for subsequent blit and line drawing functions.

Function Prototype

```
GF_RETTYPE  GFGxSetClip(
    GF_HANDLE  GxHandle,
    PGFRECT    pClipRect,
    NvU32      clipFlag );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
pClipRect	Pointer to the clipping rectangle. See "GFRECT" on page 11.
clipFlag	One of the following: • GFGX_SETCLIP_DISABLE • GFGX_SETCLIP_ENABLE • GFGX_SETCLIP_SETRECT

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Clipping Sample Code

```
/*
    Initialize clipping and rectangle parameters.
*/
clipRect.top    = 10;
clipRect.left   = 10;
clipRect.right  = 60;
clipRect.bottom = 60;
/*
    Set the clipping rectangle and enable clipping.
*/
GFGxSetClip( GxHandle, &clipRect, (GFGX_SETCLIP_SETRECT|
                                   GFGX_SETCLIP_ENABLE) );
/*
    Call drawing function: Rectangle Fill, Pattern Fill, etc.
*/
GFGXFillRect( GxHandle, 20, 20, 30, 30, 0x07E0 );
/*
    Disable clipping.
*/
GFGxSetClip( GxHandle, NULL, GFGX_SETCLIP_DISABLE );
```

Clipping Flags

```
#define GFGX_SETCLIP_DISABLE    0x00000000UL
#define GFGX_SETCLIP_ENABLE     0x00000001UL
#define GFGX_SETCLIP_SETRECT    0x00000002UL
  // Set clipping rectangle
#define GFGX_SETCLIP_INSIDE     0x00000004UL
  // Set inside clipping (default is outside clippin
```

GFGxSetPal()

This function sets new values into the palette entry at the **`index`** location. The index value ranges from `[0–255]` inclusively. Parameter **`palVal`** is in RGB format, where bits `[0-7]` represent the red value, bits `[8-15]` the green value, and bits `[16-23]` the blue value. When this function is called, the physical palette entry is modified permanently.

This function is used only for the 8-bits-per-pixel mode and does not apply to the RGB 16-bit mode.

Function Prototype

```
GF_RETTYPE  GFGxSetPal(
    GF_HANDLE  GxHandle,
    NvU32      palVal,
    NvU16      index );
```

Parameters

`GxHandle`	Handle specific to the GFGxAPI.
`palVal`	New palette entry value.
`index`	Index of the palette entry.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFGxSetPalRange()

This function sets the a range of new values pointed by **`pPalRange`** into the palette entries starting at **`startIndex`** location and ending at **`endIndex`** inclusively. The index value ranges from `[0~255]` inclusively. Each palette entry value is in RGB format, where bits `[0-7]` represent the red value, bits `[8-15]` the green value, and bits `[16-23]` the blue value. When this function is called, the physical palette entries are modified permanently.

This function is used only for the 8-bits-per-pixel mode and does not apply to the RGB 16-bit mode.

Function Prototype

```
GF_RETTYPE  GFGxSetPalRange(
    GF_HANDLE  GxHandle,
    NvU32      *pPalRange,
    NvU16      startIndex,
    NvU16      endIndex );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
*pPalRange	Pointer to a range of palette entry values.
startIndex	Starting index of palette entries.
endIndex	Ending index of palette entries.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxGetPal()

This function returns the palette entry value at **index** location. The index value ranges from [0–255] inclusively. The returned palette entry value is stored in the location pointed to by **pPalVal**.

GFGxGetPal() is used only for the 8-bits-per-pixel mode and does not apply to the RGB 16-bit mode.

Function Prototype

```
GF_RETTYPE  GFGxGetPal(
    GF_HANDLE  GxHandle,
    NvU32      *pPalVal,
    NvU16      index );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
*pPalVal	Pointer to palette entry value.
index	Index of palette entry.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxGetPalRange()

This function returns the palette entry values starting from location **startIndex** to **endIndex** inclusively. The returned values are stored into variables pointed to by **pPalRange**.

This function is used only for the 8-bits-per-pixel mode and does not apply to the RGB 16-bit mode.

Function Prototype

```
GF_RETTYPE  GFGxGetPalRange(
    GF_HANDLE  GxHandle,
    NvU32      *pPalRange,
    NvU16      startIndex,
    NvU16      endIndex );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
*pPalRange	Pointer to a range of palette entry variables.
startIndex	Starting index of palette entries.
endIndex	Ending index of palette entries.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxBltFullScreen()

This function transfers a color bitmap of the exact full screen size from system memory to video memory.

Function Prototype

```
GF_RETTYPE  GFGxBltFullScreen(GF_HANDLE     GxHandle,
                              GFGXBLTPARAM *pBltParam);
```

Parameters

GxHandle	Handle to GFGxAPI
*pBltParam	Refer to structure definition

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxWaitNotBusy()

This function checks whether the graphics engine is busy or idle. If the graphics engine is busy, the function waits until it is idle or until the timeout has expired. For example, if you want to write to the frame buffer but the graphics engine is busy, the function can be used to wait until it is idle. **`GFGxWaitNotBusy()`** blocks the CPU thread while waiting for the graphics engine to become idle. It behaves the same for all GoForce processors.

Function Prototype

```
GF_RETTYPE  GFGxWaitNotBusy(
    GF_HANDLE  GxHandle,
    NvU32      timeout );
```

Parameters

`GxHandle`	Handle specific to the GFGxAPI.
`timeout`	Time-out value in number of tries.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFGxNotBusy()

Like **`GFGxWaitNotBusy()`**, this function checks to see if the graphics engine is busy or idle, with this difference: the CPU is not blocked (does not wait) until the graphics engine is idle. Control returns to the CPU right after a wait command is placed in the graphics engine command FIFO. The GoForce hardware begins executing the commands following the engine idle command only after the graphics engine becomes idle.

This function works as described only on processors that support the 3D engine, such as the GoForce 3D 4500 and GoForce 3D 4800. For all other processors, it is implemented as the **`GFGxWaitNotBusy()`** function.

Function Prototype

```
GF_RETTYPE  GFGxNotBusy(
    GF_HANDLE  GxHandle,
    NvU32      timeout );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
timeout	Timeout value in number of tries.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxSetAttribute()

This function is used to set the bit depth, the destination stride, the start address of the display buffer, the associated surface of the display buffer, or the graphics command execution control.

Function Prototype

```
GF_RETTYPE  GFGxSetAttribute(
    GF_HANDLE  GxHandle,
    NvU32      aid,
    NvU32      attr );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
aid	Attribute ID, which is one of the following: • GFGX_ATTR_BPP Bit depth. • GFGX_ATTR_STRIDE Stride in bytes. • GFGX_ATTR_ADDR_PTR Pointer to the start address of the display buffer. • GFGX_ATTR_ADDR Offset to the start address of the display buffer. • GFGX_ATTR_SURFACE Surface for the display. • GFGX_ATTR_VSYNC Specifies when the current graphics command starts. If attr is 0, the command executes immediately. If attr is 1, the command starts on the vertical sync.
attr	Attribute value.

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFGxGetAttribute()

This function returns the bit depth, the destination stride, the start address, the associated surface of the display buffer, or the graphics command execution control. The location of the returned attribute value is pointed to by **attr**.

Function Prototype

```
GF_RETTYPE  GFGxGetAttribute(
    GF_HANDLE  GxHandle,
    NvU32      aid,
    NvU32      *attr );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
aid	Attribute ID. See "GFGxSetAttribute()" on page 183.
*attr	Pointer to attribute value.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxReadBlt()

Copies the bitmap from the screen to system memory.

Function Prototype

```
GF_RETTYPE  GFGxReadBlt(
    GF_HANDLE  GxHandle,
    NvS16      dx,
    NvS16      dy,
    NvS16      w,
    NvS16      h,
    NvS16      sx,
    NvS16      sy,
```

Function Prototype

```
    NvU16       dstStride,
    NvU8        *pDstAddr );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
dx	Destination *x* in pixels (relative to left).
dy	Destination *y* in scan lines (relative to top).
w	Width of the rectangle in pixels.
h	Height of the rectangle in scan lines.
sx	Source *x* in pixels (relative to left of source bitmap).
sy	Source *y* in scan lines (relative to top of source bitmap).
dstStride	Destination stride in bytes.
*pDstAddr	Pointer to destination address in system memory.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxStretchBlt()

GFGxStretchBlt() is used to stretch or shrink an image, which can be in system memory or video memory. The function has these limitations:

- Supports only 16-bpp color depth.
- Does not support stretching the image in the x direction while shrinking it in the y direction, or vice versa. It only supports stretch-stretch and shrink-shrink cases.

Function Prototype

```
GF_RETTYPE  GFGxBlt(
    GF_HANDLE     GxHandle,
    GFGXBLTPARAM  *pBltParam );
```

Parameters

GxHandle	Handle specific to the GFGxAPI.
*pBltParam	See "GFGxAPI Data Structure: GFGXBLTPARAM" on page 203.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxFillRectEx()

This function fills a rectangle in video memory with the specified solid color.

Function Prototype

```
GF_RETTYPE  GFGxFillRectEx(GF_HANDLE GxHandle,
                           PGFRMSURFACE pSurf,
                           NvS16 x, NvS16 y,
                           NvS16 w, NvS16 h,
                           NvU32 color);
```

Parameters

GxHandle	Handle to GFGxAPI
x,y	Left-top corner of the rectangle
w,h	Width & height of the rectangle
color	Ccolor to be filled (8BPP index, or 16BPP-32BPP TrueColor)

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyRectEx()

This function copies a rectangular image in the video memory to another location in the same video memory. It is commonly used in scrolling an image in video memory.

The video memory location is specified by the surface parameter.

GFGxCopyRect() checks for the destruction of the copied image in video memory caused by overlapping source and destination locations. Also, if the destination for the rectangle exceeds the limit of the video memory, wrapping is done in hardware and no error code is returned.

Function Prototype

```
GF_RETTYPE  GFGxCopyRectEx(GF_HANDLE GxHandle,
                           PGFRMSURFACE pSurf,
                           NvS16 dx, NvS16 dy,
                           NvS16 w, NvS16 h,
                           NvS16 sx, NvS16 sy);
```

Parameters

GxHandle	Handle to GFGxAPI
pSurf	Pointer to destination surface
dx,dy	Left-top corner of destination rectangle
w,h	Width & height of rectangles
sx,sy	Left-top corner of source rectangle

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyMonoBitmapEx()

This function copies a source bitmap from system memory to the video memory.

The source bitmap is specified in monochrome (1BPP) pixels.

Video memory location is specified by the surface parameter.

Function Prototype

```
GF_RETTYPE  GFGxCopyMonoBitmapEx( GF_HANDLE GxHandle,
                                  PGFRMSURFACE pSurf,
                                  NvS16 dx, NvS16 dy,
                                  NvS16 w, NvS16 h,
                                  NvS16 sx, NvS16 sy,
                                  NvU32 fgColor,
                                  NvU32 bgColor,
                                  NvS16 srcStride,
                                  NvU8 *pMonoBits);
```

Parameters

GxHandle	Handle to GFGxAPI
pSurf	Pointer to destination surface
dx,dy	Left-top corner of destination rectangle
w,h	Width & height of rectangles
sx,sy	Left-top corner of source rectangle
fgColor	Color applied to source data bit 1
bgColor	Color applied to source data bit 0
sbgColor	Source stride in bytes
pMonoBits	Pointer to monochome (1bpp) bitmap

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyTransMonoBitmapEx()

This function copies a source bitmap from system memory to the video memory.

The source bitmap is specified in monochrome (1BPP) pixels and can be transparent. Video memory location is specified by the surface parameter.

This function is similar to GFGxCopyMonoBitmapEx() except for the select Transparameter. This parameter indicates whether bit data equal to 0 or 1 is going to be transparent.

This function could be used for drawing a transparent text string or character.

Function Prototype

```
GF_RETTYPE  GFGxCopyTransMonoBitmapEx(
              GF_HANDLE GxHandle,
              PGFRMSURFACE pSurf,
              NvS16 dx, NvS16 dy,
              NvS16 w, NvS16 h,
              NvS16 sx, NvS16 sy,
              NvU32 color,
              NvS16 srcStride,
              NvU8 *pMonoBits,
              NvS16 selectTrans);
```

Parameters

GxHandle	Handle to GFGxAPI
pSurf	Pointer to destination surface
dx,dy	Left-top corner of destination rectangle
w,h	Width & height of the rectangle in pixels and scan lines
sx,sy	Left-top corner of source rectangle
color	Color applied to non-transparent source data bit
srcStride	Source stride in bytes
pMonoBits	Pointer to monochome (1bpp) bitmap
selectTrans	Transparency select: 0 or 1

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyColorBitmapEx()

This function copies a source bitmap from system memory to the video memory.

The source bitmap's color depth must match the destination's video memory's color depth.

Video memory location is specified by the surface parameter.

Function Prototype

```
GF_RETTYPE GFGxCopyColorBitmapEx( GF_HANDLE GxHandle,
                                  PGFRMSURFACE pSurf,
                                  NvS16 dx, NvS16 dy,
                                  NvS16 w, NvS16 h,
                                  NvS16 sx, NvS16 sy,
                                  NvS16 srcStride,
                                  NvU8  *pColorBits
                                  );
```

Parameters

GxHandle	Handle to GFGxAPI
pSurf	Pointer to destination surface
dx,dy	Left-top corner of destination rectangle

Parameters (continued)

w,h	Width & height of the rectangle in pixels and scan lines
sx,sy	Left-top corner of source rectangle
srcStride	Source stride in bytes
pColorBits	Pointer to color bitmap

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxLineEx()

This function draws a line from a point x1, y1 to another point x2, y2 with a specified color and ROP2 code.

Video memory location is specified by the surface parameter.

Function Prototype

```
    GF_RETTYPE  GFGxLineEx( GF_HANDLE GxHandle,
                            PGFRMSURFACE pSurf,
                            NvU16 x1, NvU16 y1,
                            NvU16 x2, NvU16 y2,
                            NvU16 rop2, NvU32 color,
                            NvU16 flags);
```

Parameters

GxHandle	Handle to GFGxAPI
pSurf	Pointer to destination surface
x1,y1	(x,y) destination point coordinates
x2,y2	(x,y) source point coordinatess
rop2	ROP2 code
color	Color key for transparent blt or pattern color
flags	Refer to GFGX_LINE_X flag definitions

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxBltFullScreenEx()

This function transfers a color bitmap of the exact full screen size from system memory to video memory.

Function Prototype

```
GF_RETTYPE  GFGxBltFullScreenEx(GF_HANDLE GxHandle,
                                PGFRMSURFACE pSurf,
                                GFGXBLTPARAM *pBltParam);
```

Parameters

GxHandle	Handle to GFGxAPI
pSurf	Pointer to destination surface
*pBltParam	Refer to structure definition

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyRectDirectEx()

This function copies a rectangular image in the video memory to another location in the same video memory without taking care of overlapping cases.

Video memory location is specified by the surface parameter.

This is a special case of GxCopyRect(). It does not check on destructions of copied image on video memory due to overlapping source and destination locations.

Function Prototype

```
GF_RETTYPE  GFGxCopyRectDirectEx(GF_HANDLE GxHandle,
                                 PGFRMSURFACE pSurf
                                 NvS16 dx, NvS16 dy,
                                 NvS16 w, NvS16 h,
                                 NvS16 sx, NvS16 sy);
```

Parameters

GxHandle	Handle to GFGxAPI
pSurf	Pointer to the destination surface
dx,dy	Left-top corner of destination rectangle

Parameters (continued)

w,h	Width & height of rectangles
sx,sy	Left-top corner of source rectangle

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyPackedColorBitmapEx()

This function copies a source bitmap from system memory to the video memory with certain restrictions.

Video memory location is specified by the surface parameter.

This is a special case of GxCopyColorBitmap(). The conditions below must be verified:

- The source bitmap pointer, pColorBits, must be a 32-bit (DWORD) aligned address.
- The size of each line of the source bitmap must be 64-bits (8 bytes, QWORD) aligned.
- There must not be any skipping bytes between lines.
- The total bitmap size must be smaller than or equal to 2K (2048) bytes.
- The source bitmap's color depth MUST match the destination video memory's color depth.

Function Prototype

```
GF_RETTYPE GFGxCopyPackedColorBitmapEx(
                    GF_HANDLE GxHandle,
                    PGFRMSURFACE pSurf,
                    NvU32 *pColorBits,
                    NvU16 bitmapSize,
                    GFRECT rect );
```

Parameters

GxHandle	Handle to GFGxAPI
pSurf	Pointer to destination surface
pColorBits	Pointer to the color bitmap

Parameters (continued)

bitmapSize	The total bytes to transfer
rect	The destination rectangle

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxBltSurfaceEx()

This function performs a surface-to-surface blit. The color depth of the source surface must match that of the destination. Also, the width and height of the source must be the same as the destination's width and height.

Alpha-blending parameters can be specified.

Function Prototype

```
GF_RETTYPE  GFGxBltSurfaceEx(
                    GF_HANDLE GxHandle,
                    PGFRMSURFACE *ppDestSurf,
                    PGFRMSURFACE *ppSrcSurf,
                    NvS16 w,
                    NvS16 h,
                    NvU16 rop3,
                    NvU32 srcStartAddr,
                    NvU32 dstStartAddr,
                    NvU32 flags,
                    NvU32 flagex1,
                    NvU32 colorCompare,
                    NvU16 alpha,
                    NvU16 alphaBg);
```

Parameters

GxHandle	Handle to GFGxAPI
*ppDestSurf	Pointer to destination surface pointer
*ppSrcSurf	Pointer so source surface pointer
w, h	Width & height of the rectangle in pixels and scan lines
rop3	Raster operation code

Parameters (continued)

srcStartAddr	Start address relative to the source surface
dstStartAddr	Start address relative to the destination surface
flags	Refer to GFGX_BLT_X flag definitions
flagex1	Refer to alphablending flag definitions GFGXEX1_BLT_X
colorCompare	Color key for transparent blit
alpha	Alpha value for GFGX_BLT_ALPHA_FIXED or alphaFg
alphaBg	Background alpha

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxFastRotate()

This function is used to transform a given source rectangle to the given destination location in the video memory.

The destination rectangle can be same than source rectangle or different.

Source and destination must be in video memory.

This function can be used the same way than GFGxBlt, adding flagex2 information. The following transformations are supported :

#GFGXEX2_FAST_ROTATE_FLIP_X	FLIP-X.
#GFGXEX2_FAST_ROTATE_FLIP_Y	FLIP-Y.
#GFGXEX2_FAST_ROTATE_TRANS_LR	TRANSPOSE LEFT-RIGHT
#GFGXEX2_FAST_ROTATE_TRANS_RL	TRANSPOSE RIGHT-LEFT
#GFGXEX2_FAST_ROTATE_ROT_90	90° rotation.
#GFGXEX2_FAST_ROTATE_ROT_180	180° rotation.
#GFGXEX2_FAST_ROTATE_ROT_270	270° rotation.

Function Prototype

```
GF_RETTYPE  GFGxFastRotate(GF_HANDLE     GxHandle,
                           GFGXBLTPARAM *pBltParam);
```

Parameters

GxHandle	Handle to GFGxAPI
*pBltParam	Refer to structure definition

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Fast Rotate Defines

```
/**** FAST ROTATION FLAGS ****/
/** GFGxAPI BITBLT Fast Rotation mask.*/
#define GFGXEX2_MASK    0x3FFFFFFF

/* Fast Rotation Modes */

/* Square (in place rotation) */
/** GFGxAPI BITBLT Fast Rotation flags : dst is same than src.
*/
#define GFGXEX2_FAST_ROTATE_SQUARE  (0x80000000 | 0x00000001UL)

/* Source to Destination Rotation ( 2 buffers) */
/** GFGxAPI BITBLT Fast Rotation flags : dst is different than
src. */
#define GFGXEX2_FAST_ROTATE_SRC_DST_COPY    (0x80000000 |
0x00000002UL)

/* Disable Fast Rotation */
/** GFGxAPI BITBLT Fast Rotation flags : disable fast rotation.
*/
#define GFGXEX2_FAST_ROTATE_DISABLE (0x80000000 | 0x00000004UL)

/* Fast Rotation Types */

/* Flip-X */
/** GFGxAPI BITBLT Fast Rotation flags : FLIP-X transformation.
*/
#define GFGXEX2_FAST_ROTATE_FLIP_X  (0x80000000 | 0x00000008UL)

/* Flip-Y */
```

Fast Rotate Defines

```
/** GFGxAPI BITBLT Fast Rotation flags : FLIP-Y transformation.
*/
#define GFGXEX2_FAST_ROTATE_FLIP_Y  (0x80000000 | 0x00000010UL)

/* Trans-LR */
/** GFGxAPI BITBLT Fast Rotation flags : TRANSPOSE LEFT-RIGHT
transformation. */
#define GFGXEX2_FAST_ROTATE_TRANS_LR    (0x80000000 |
0x00000020UL)

/* Trans-RL */
/** GFGxAPI BITBLT Fast Rotation flags : TRANSPOSE RIGHT-LEFT
transformation. */
#define GFGXEX2_FAST_ROTATE_TRANS_RL    (0x80000000 |
0x00000040UL)

/* 90 deg Rotation */
/** GFGxAPI BITBLT Fast Rotation flags : 90° rotation. */
#define GFGXEX2_FAST_ROTATE_ROT_90  (0x80000000 | 0x00000080UL)

/* 180 deg Rotation */
/** GFGxAPI BITBLT Fast Rotation flags : 180° rotation. */
#define GFGXEX2_FAST_ROTATE_ROT_180 (0x80000000 | 0x00000100UL)

/* 270 deg Rotation */
/** GFGxAPI BITBLT Fast Rotation flags : 270° rotation. */
#define GFGXEX2_FAST_ROTATE_ROT_270 (0x80000000 | 0x00000200UL)

/* Identity */
/** GFGxAPI BITBLT Fast Rotation flags : identity
transformation. */
#define GFGXEX2_FAST_ROTATE_IDENTITY    (0x80000000 |
0x00000400UL)
```

GFGxCopyTransColorBitmap()

This function copies a source bitmap from system memory to the video memory.

The source bitmap's color depth must match the destination's video memory's color depth.

Source transparency is supported.

Function Prototype

```
GF_RETTYPE GFGxCopyTransColorBitmap(
                GF_HANDLE GxHandle,
                NvS16     dx,
                NvS16     dy,
                NvS16     w,
                NvS16     h,
                NvS16     sx,
                NvS16     sy,
                NvS16     srcStride,
                NvU8      *pColorBits,
                NvU32     *pSrcColorKey);
```

Parameters

GxHandle	Handle to GFGxAPI
dx,dy	Left-top corner of destination rectangle
w,h	Width & height of the rectangle in pixels and scan lines
sx,sy	Left-top corner of source rectangle
srcStride	Source stride in bytes
pColorBits	Pointer to color bitmap
pSrcColorKey	Pointer to source color key. If NULL no transparency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyTransColorBitmapWithMonoPattern()

This function copies a source bitmap from system memory to the video memory with a mono pattern buffer combination.

The source bitmap's color depth must match the destination's video memory's color depth.

Source and mono pattern transparencies are supported. Depending of pattern size, replicated mode is used or not.

Function Prototype

```
GF_RETTYPE GFGxCopyTransColorBitmapWithMonoPattern(
                GF_HANDLE GxHandle,
                NvS16     dx,
                NvS16     dy,
                NvS16     w,
                NvS16     h,
                NvS16     sx,
                NvS16     sy,
                NvS16     srcStride,
                NvU8      *pColorBits,
                NvU32     *pSrcColorKey,
                NvU8      rop3,
                NvU8      *pPatBuffer,
                NvU32     patFgColor,
                NvU32     patBgColor,
                NvU32     patW,
                NvU32     patH,
                NvU32     patStride,
                NvU16     *pSelectPatTrans);
```

Parameters

GxHandle	Handle to GFGxAPI
dx,dy	Left-top corner of destination rectangle
w,h	Width & height of the rectangle in pixels and scan lines
sx,sy	Left-top corner of source rectangle
srcStride	Source stride in bytes
pColorBits	Pointer to color bitmap
pSrcColorKey	Pointer to source color key. If NULL no transparency.

Parameters (continued)

rop3	Raster operation code
pPatBuffer	Pointer to pattern buffer
patFgColor	Pattern foreground color
patBgColor	Pattern background color
patW, patH	Wwidth & height of pattern buffer
patStride	Pattern stride in bytes
pSelectPatTrans	Pointer to pattern color key. If NULL no transparency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyTransColorBitmapWithColorPattern()

This function copies a source bitmap from system memory to the video memory with a color pattern buffer combination.

The source bitmap and pattern's color depth must match the destination's video memory's color depth.

Source and color pattern transparencies are supported.

Function Prototype

```
GF_RETTYPE GFGxCopyTransColorBitmapWithColorPattern(
                GF_HANDLE GxHandle,
                NvS16     dx,
                NvS16     dy,
                NvS16     w,
                NvS16     h,
                NvS16     sx,
                NvS16     sy,
                NvS16     srcStride,
                NvU8      *pColorBits,
                NvU32     *pSrcColorKey,
                NvU8      rop3,
                NvU8      *pPatBuffer,
                NvU32     patW,
                NvU32     patH,
```

Function Prototype

```
                    NvU32     patStride,
                    NvU32     *pPatColorKey);
```

Parameters

GxHandle	Handle to GFGxAPI
dx,dy	Left-top corner of destination rectangle
w,h	Width & height of the rectangle in pixels and scan lines
sx,sy	Left-top corner of source rectangle
srcStride	Source stride in bytes
pColorBits	Pointer to color bitmap
pSrcColorKey	Pointer to source color key. If NULL no transparency.
rop3	Raster operation code
pPatBuffer	Pointer to pattern buffer
patW, patH	Width & height of pattern buffer
patStride	Pattern stride in bytes
pPatColorKey	Ppointer to pattern color key. If NULL no transparency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyTransMonoBitmapWithMonoPattern()

This function copies a source bitmap from system memory to the video memory with a mono pattern buffer combination.

The source bitmap and pattern are mono (1BPP).

Mono source and mono pattern transparencies are supported. Depending on pattern size, replicated mode is used or not.

Function Prototype

```
    GF_RETTYPE GFGxCopyTransMonoBitmapWithMonoPattern(
                    GF_HANDLE GxHandle,
                    NvS16     dx,
                    NvS16     dy,
                    NvS16     w,
                    NvS16     h,
```

Function Prototype

```
                    NvS16     sx,
                    NvS16     sy,
                    NvU32     srcFgColor,
                    NvU32     srcBgColor,
                    NvS16     srcStride,
                    NvU8      *pMonoBits,
                    NvU16     *pSelectSrcTrans,
                    NvU8      rop3,
                    NvU8      *pPatBuffer,
                    NvU32     patFgColor,
                    NvU32     patBgColor,
                    NvU32     patW,
                    NvU32     patH,
                    NvU32     patStride,
                    NvU16     *pSelectPatTrans);
```

Parameters

`GxHandle`	Handle to GFGxAPI
`dx, dy`	Left-top corner of destination rectangle
`w, h`	Width & height of the rectangle in pixels and scan lines
`sx, sy`	Left-top corner of source rectangle
`srcFgColor`	Source foreground color
`srcBgColor`	Source background color
`srcStride`	Source stride in bytes
`pMonoBits`	Pointer to source bitmap
`pSelectSrcTrans`	Pointer to source color key. If NULL no transparency.
`rop3`	Raster operation code
`pPatBuffer`	Pointer to pattern buffer
`patFgColor`	Pattern foreground color
`patBgColor`	Pattern background color
`patW, patH`	Width & height of pattern buffer
`patStride`	Pattern stride in bytes
`pSelectPatTrans`	Pointer to pattern color key. If NULL no transparency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxCopyTransMonoBitmapWithColorPattern()

This function copies a source bitmap from system memory to the video memory with a color pattern buffer combination.

The source bitmap is mono (1BPP).

Mono source and color pattern transparencies are supported.

Function Prototype

```
GF_RETTYPE GFGxCopyTransMonoBitmapWithColorPattern(
                GF_HANDLE GxHandle,
                NvS16     dx,
                NvS16     dy,
                NvS16     w,
                NvS16     h,
                NvS16     sx,
                NvS16     sy,
                NvU32     srcFgColor,
                NvU32     srcBgColor,
                NvS16     srcStride,
                NvU8      *pMonoBits,
                NvU16     *pSelectSrcTrans,
                NvU8      rop3,
                NvU8      *pPatBuffer,
                NvU32     patW,
                NvU32     patH,
                NvU32     patStride,
                NvU32     *pPatColorKey);
```

Parameters

GxHandle	Handle to GFGxAPI
dx, dy	Left-top corner of destination rectangle
w, h	Width & height of the rectangle in pixels and scan lines
sx, sy	Left-top corner of source rectangle
srcFgColor	Source foreground color

Parameters (continued)

srcBgColor	Source background color
srcStride	Source stride in bytes
pMonoBits	Pointer to source bitmap
pSelectSrcTrans	Pointer to source color key. If NULL no transparency.
rop3	Raster operation code
pPatBuffer	Pointer to pattern buffer
patW, patH	Width & height of pattern buffer
patStride	Pattern stride in bytes
pPatColorKey	Pointer to pattern color key. If NULL no transparency.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFGxAPI Data Structure: GFGXBLTPARAM

This is the parameter for the **GFGxBlt()** function. It is a "super" structure that contains almost all the information required for a blit operation. See detailed descriptions for specifying this structure at "GFGxBlt()" on page 165

GFGXBLTPARAM Structure

```
typedef struct _GFGXBLTPARAM
{
    NvS16               dx;
    NvS16               dy;
    NvS16               w;
    NvS16               h;
    NvS16               sx;
    NvS16               sy;
    NvS16               sw;
    NvS16               sh;
    NvU16               bpp;
    NvS16               stride;
    NvU32               fgColor;
    NvU32               bgColor;
    NvU8                *pBits;
```

GFGXBLTPARAM Structure (continued)

```
    NvS16               px;
    NvS16               py;
    NvU32               patFgColor;
    NvU32               patBgColor;
    NvU32               patColorCmp;
    NvU8                *pPatBits;
    NvU32              patStride;
    NvU32              patW;
    NvU32              patH;
    NvU32              patID;

    // Miscellaneous parameters
    NvU32               colorCompare;
    NvU32               flags;
    NvU16               rop3;

    // Surface parameters
    PGFRMSURFACE        *ppDestSurf;
    PGFRMSURFACE        *ppSrcSurf;
    NvU32               srcStartAddr;
    NvU32               dstStartAddr;
    NvS16               dstStride;

    // alpha blending parameters
    NvU16               alpha;
    NvU16               alphaBg;
    NvU8                *pAlphaplane;

    // Fading parameters
    NvU32               fadeCoeff;
    NvU32               fadeOffset;

    /** Alphablending flag. */
    NvU32   flagex1;

    /** Fast rotation flag. */
    NvU32   flagex2;
```

GFGXBLTPARAM Structure (continued)

```
} GFGXBLTPARAM, *PGFGXBLTPARAM;
```

GFGXBLTPARAM Fields

Destination Location and Dimension Parameters	
dx	Destination *x* in pixels (relative to left).
dy	Destination *y* in scan lines (relative to top).
w	Width of the rectangle in pixels.
h	Height of the rectangle in scan lines.
Source Bitmap Parameters	
sx	Source *x* in pixels (relative to left).
sy	Source *y* in scan lines (relative to top).
sw	Width of the source rectangle in pixels.
sh	Height of the source rectangle in scan lines.
bpp	Color depth.
stride	Number of bytes per scan line for bitmap.
fgColor	Foreground color (used if pixel data is 1 on 1-bpp source bitmap).
bgColor	Background color (used if pixel data is 0 on 1-bpp source bitmap).
***pBits**	Pointer to source bitmap.
Pattern Bitmap Parameters	
px	Pattern offset in x direction.
py	Pattern offset in y direction.
patFgColor	Foreground color (used if pixel data is 1 on 1-bpp source bitmap).
patBgColor	Background color (used if pixel data is 0 on 1-bpp source bitmap).
Miscellaneous Parameters	
patColorCmp	Pattern color key for transparent blt.
pPatBits	Pointer to pattern bitmap.
patStride	Number of bytes per scanline for pattern.
patW	Width of pattern.
patH	Height of pattern.
patID	Pattern ID when cache is used.
colorCompare	Color key for transparent blit.

GFGXBLTPARAM Fields (continued)

flags	See GFGXBLTPARAM definitions below.
rop3	ROP3 code.
Surface, Linear Mode, and Destination Stride Parameters	
***ppDestSurf**	Pointer to destination surface pointer.
***ppSrcSurf**	Pointer to source surface pointer.
srcStartAddr	Source start address offset.
dstStartAddr	Destination start address offset.
dstStride	Destination stride. Should be specified only if the present stride is different from the required destination stride. Flag GFGX_BLT_SET_DST_STRIDE must be specified.
Alpha Blending Parameters	
alpha	Alpha value for fixed alpha blending, or foreground alpha value for source 1555 alpha blending.
alphaBg	Background alpha value for source 1555 alpha blending.
Fading Parameters	
fadeCoeff	Fading coefficient.
fadeOffset	Fading offset.
flagex1	Alphablending flag.
flagex2	Fast rotation flag.

GFGXBLTPARAM Definitions

```
/**** BITBLT - SOURCE FLAGS ****/

/** GFGxAPI BITBLT Source flags : source comes from video
memory.*/
#define GFGX_BLT_SRC_VIDEO        0x00000000UL

/** GFGxAPI BITBLT Source flags : colour source comes from
system memory. */
#define GFGX_BLT_SRC_SYSMEM_COLOR   0x00000001UL

/** GFGxAPI BITBLT Source flags : mono source comes from system
mem. */
#define GFGX_BLT_SRC_SYSMEM_MONO    0x00000002UL
```

GFGXBLTPARAM Definitions

```
/** GFGxAPI BITBLT Source flags : source is solid (unique
color). */
#define GFGX_BLT_SRC_SOLID          0x00000004UL


/**** BITBLT - PATTERN FLAGS ****/

/** GFGxAPI BITBLT Pattern flags : pattern is mono. */
#define GFGX_BLT_PAT_MONO           0x00000008UL

/** GFGxAPI BITBLT Pattern flags : pattern is color. */
#define GFGX_BLT_PAT_COLOR          0x00000010UL

/** GFGxAPI BITBLT Pattern flags : pattern is solid (unique
color).*/
#define GFGX_BLT_PAT_SOLID          0x00000020UL

/** GFGxAPI BITBLT Pattern flags : pattern is cached in video
mem. */
#define GFGX_BLT_PAT_CACHE          0x00000040UL


/**** BITBLT - MODE FLAGS ****/
/**** Linear / XY Mode / Surface Flag ****/

/** GFGxAPI BITBLT Mode flags : X/Y mode is used.*/
#define GFGX_BLT_MODE_XY            0x00000000UL

/** GFGxAPI BITBLT Mode flags : linear mode is used. */
#define GFGX_BLT_MODE_LINEAR        0x00000080UL

/** GFGxAPI BITBLT Mode flags : surface to surface mode is
used. */
#define GFGX_BLT_SURFACE           0x00000100UL


/**** BITBLT - CLIPPING FLAGS ****/
```

GFGXBLTPARAM Definitions

```
/** GFGxAPI BITBLT Clipping flags : enable clipping. */
#define GFGX_BLT_CLIP              0x00000200UL

/**** BITBLT - SOURCE TRANSPARENCY FLAGS ****/

/** GFGxAPI BITBLT Source Transparency flags : enable color
source transparency     (if source pixel matches colorCompare,
do not overwrite).*/
#define GFGX_BLT_TRANSPARENT_SRC_COLOR  0x00000400UL

/** GFGxAPI BITBLT Source Transparency flags : enable color
source inverse
    transparency (if source pixel matches colorCompare,
overwrite).*/
#define GFGX_BLT_TRANSPARENT_SRC_COLOR_INV 0x08000000UL


/** GFGxAPI BITBLT Source Transparency flags : enable mono
source transparency (if source bit is 0, no overwrite on
corresponding destination location).*/
#define GFGX_BLT_TRANSPARENT_SRC_MONO   0x00000800UL

/** GFGxAPI BITBLT Source Transparency flags : enable mono
source inverse     transparency (if source bit is 1, no
overwrite on corresponding destination     location).*/
#define GFGX_BLT_TRANSPARENT_SRC_MONO_INV   0x00001000UL


/**** BITBLT - DESTINATION TRANSPARENCY FLAGS ****/

/** GFGxAPI BITBLT Destination Transparency flags : enable
destination     transparency (if destination pixel matches
colorCompare, overwrite).*/
#define GFGX_BLT_TRANSPARENT_DST    0x00002000UL

/** GFGxAPI BITBLT Destination Transparency flags : enable
destination inverse transparency (if destination pixel matches
colorCompare, do not overwrite).*/
```

GFGXBLTPARAM Definitions

```
#define GFGX_BLT_TRANSPARENT_DST_INV    0x10000000UL

/**** BITBLT - PATTERN TRANSPARENCY FLAGS ****/

/** GFGxAPI BITBLT Pattern Transparency flags : enable mono
pattern transparency(if pattern bit is 0, no overwrite on
corresponding destination location).*/
#define GFGX_BLT_TRANSPARENT_PAT_MONO   0x00004000UL

//** GFGxAPI BITBLT Pattern Transparency flags : enable mono
pattern inverse transparency (if pattern bit is 1, no overwrite
on corresponding destination location).*/
#define GFGX_BLT_TRANSPARENT_PAT_MONO_INV   0x00008000UL

/** GFGxAPI BITBLT Pattern Transparency flags : enable color
pattern transparency (if pattern pixel matches colorCompare, do
not overwrite).*/
#define GFGX_BLT_TRANSPARENT_PAT_COLOR  0x00010000UL

//** GFGxAPI BITBLT Pattern Transparency flags : enable color
pattern inverse transparency (if pattern pixel matches
colorCompare, overwrite). */
#define GFGX_BLT_TRANSPARENT_PAT_COLOR_INV  0x20000000UL


/**** Fade Blt ****/
/** GFGxAPI BITBLT flags : enable fade blt. */
#define GFGX_BLT_FADE_BLT   0x00020000UL

/**** Mask Blt ****/
/** GFGxAPI BITBLT flags : enable mask blt. */
#define GFGX_BLT_MASK_BLT   0x00040000UL

/** GFGxAPI BITBLT flags : enable alphablending. */
#define GFGX_BLT_ALPHA_BLENDING 0x00080000UL

/**** set Destination stride ****/
/** GFGxAPI BITBLT flags : set the destination stride. */
#define GFGX_BLT_SET_DST_STRIDE 0x00100000UL
```

GFGXBLTPARAM Definitions

```
/**** ALPHABLENDING FLAGS ****/

/** GFGxAPI BITBLT Alphablending flags : alpha mode is fixed
value.   */
#define GFGX_BLT_ALPHA_FIXED    ( 0x00200000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
fixed value. */
#define GFGX_BLT_ALPHA_FIXED_INV    (0x00400000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is source
ARGB1555. */
#define GFGX_BLT_ALPHA_SRC_1555_T   (0x00800000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
source ARGB1555. */
#define GFGX_BLT_ALPHA_SRC_1555_T_INV    (0x01000000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is source
ARGB4444. */
#define GFGX_BLT_ALPHA_SRC_4444 (0x02000000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
source ARGB4444. */
#define GFGX_BLT_ALPHA_SRC_4444_INV (0x04000000UL)

/** GFGxAPI BITBLT Alphablending mask.*/
#define GFGXEX1_MASK    0x3FFFFFFF

/* Fixed alpha operation */
/** GFGxAPI BITBLT Alphablending flags : alpha mode is fixed
value. */
#define GFGXEX1_BLT_ALPHA_FIXED    (0x40000000 | 0x00000001UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
fixed value. */
#define GFGXEX1_BLT_ALPHA_FIXED_INV    (0x40000000 |
0x00000002UL)
```

GFGXBLTPARAM Definitions

```
/** GFGxAPI BITBLT Alphablending flags : alpha mode is source
ARGB1555. */
#define GFGXEX1_BLT_ALPHA_SRC_1555_T   (0x40000000 |
0x00000004UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
source ARGB1555. */
#define GFGXEX1_BLT_ALPHA_SRC_1555_T_INV    (0x40000000 |
0x00000008UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is source
ARGB4444. */
#define GFGXEX1_BLT_ALPHA_SRC_4444 (0x40000000 | 0x00000010UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
source ARGB4444. */
#define GFGXEX1_BLT_ALPHA_SRC_4444_INV (0x40000000 |
0x00000020UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is source
ARGB8888. */
#define GFGXEX1_BLT_ALPHA_SRC_8888 (0x40000000 |  0x00000040UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
source ARGB8888. */
#define GFGXEX1_BLT_ALPHA_SRC_8888_INV (0x40000000 |
0x00000080UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is
destination ARGB8888. */
#define GFGXEX1_BLT_ALPHA_DST_8888 (0x40000000 | 0x00000100UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
destination ARGB8888.*/
#define GFGXEX1_BLT_ALPHA_DST_8888_INV (0x40000000 |
0x00000200UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is 1BPP
planar. */
```

GFGXBLTPARAM Definitions

```
#define GFGXEX1_BLT_PLANAR_1BPP    (0x40000000 |  0x00000400UL
)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
1BPP planar.  */
#define GFGXEX1_BLT_PLANAR_1BPP_INV    (0x40000000 |
0x00000800UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is 2BPP
planar. */
#define GFGXEX1_BLT_PLANAR_2BPP    (0x40000000 | 0x00001000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
2BPP planar. */
#define GFGXEX1_BLT_PLANAR_2BPP_INV    (0x40000000 |
0x00002000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is 4BPP
planar. */
#define GFGXEX1_BLT_PLANAR_4BPP    (0x40000000 |  0x00004000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
4BPP planar.  */
#define GFGXEX1_BLT_PLANAR_4BPP_INV    (0x40000000 |
0x00008000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is 8BPP
planar. */
#define GFGXEX1_BLT_PLANAR_8BPP    (0x40000000 | 0x00010000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
8BPP planar. */
#define GFGXEX1_BLT_PLANAR_8BPP_INV    (0x40000000 |
0x00020000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is
SRC4DST4BPP planar. */
#define GFGXEX1_BLT_PLANAR_44BPP   (0x40000000 | 0x00040000UL)
```

GFGXBLTPARAM Definitions

```
/** GFGxAPI BITBLT Alphablending flags : alpha mode is 32BPP
src to 16BPP dst. */
#define GFGXEX1_BLT_PLANAR_32BPP16 (0x40000000 | 0x00080000UL)

/** GFGxAPI BITBLT Alphablending flags : alpha mode is inverse
32BPP src to 16BPP dst. */
#define GFGXEX1_BLT_PLANAR_32BPP16_INV (0x40000000 |
0x00100000UL)
```

Note that some flag definitions follow a precedence order. If flag bits that conflict with one another are specified, the flag bit with the higher precedence is processed. The lower precedence flag is discarded, and no error is generated. The following table shows the precedence pairings.

Higher Precedence	Lower Precedence
GFGX_BLT_TRANSPARENT_SRC_COLOR	GFGX_BLT_TRANSPARENT _DST
GFGX_BLT_TRANSPARENT_SRC_COLOR	GFGX_BLT_TRANSPARENT_SRC_MONO_***
GFGX_BLT_TRANSPARENT_SRC_MONO	GFGX_BLT_TRANSPARENT_SRC_MONO_INV
GFGX_BLT_SRC_SYSMEM_COLOR	GFGX_BLT_SRC_SYSMEM_MONO

GFGxAPI Programming Sequence

1. Make sure that the build project or file has the right paths to the library and header files. Use relative directory addressing if possible. The necessary include directories are in **`GFSDK\Inc`**.
2. Include GF.h and GFGx.h in your source file. These headers are all you need to access the exposed GFSDK functions.
3. Make sure you have a resource manager handle obtained through **`GFRmOpen()`**.
4. Call **`GFRmComponentGet()`** with **`ComponentType`** set to **`GF_GXAPI`** before calling any GFGxAPI functions. Use the **`GF_HANDLE`** returned from this call for all subsequent GFGxAPI functions.
5. To better utilize the GFGxAPI functions, it's a good idea to call **`GFGxGetProperty()`** to determine the version that you are using.
6. Make calls to the appropriate GFGxAPI functions. Always remember to pass the same **`GF_HANDLE`** returned from **`GFRmComponentGet()`**.

7. When the application exits, **`GFRmComponentRelease()`** must be called with the **`GF_HANDLE`** obtained through **`GFRmComponentGet()`** to release the GFGxAPI resources.

Raster Operation (ROP) Codes

The tables below list the meanings of operands, bit-wise operators, and the complete 16 ROP2 and 256 ROP3 codes. These codes are all supported by NVIDIA media processors and are also compatible with the Microsoft Windows graphics API specification. Note that Reverse Polish notation is used to describe each code's operation. See the *Interpretations* in the ROP2 table for examples of deciphering Reverse Polish notation.

Operands

D Destination bitmap.

P Pattern data (only for ROP3).

S Source bitmap.

Bit-wise Operators

a AND

n NOT

o OR

x XOR

ROP2 Codes

ROP2 (Hex)	Equivalent ROP3 (Hex)	Reverse Polish	Interpretation
0	00	0	Fill Dest with 0
1	11	DSon	NOT (Dest OR Src)
2	22	DSna	Dest AND (NOT Src)
3	33	Sn	NOT Src
4	44	SDna	Src AND (NOT Dest)
5	55	Dn	NOT Dest
6	66	DSx	Dest XOR Src
7	77	DSan	NOT (Dest AND Src)

ROP2 Codes (continued)

ROP2 (Hex)	Equivalent ROP3 (Hex)	Reverse Polish	Interpretation
8	88	DSa	Dest AND Src
9	99	DSxn	NOT (Dest XOR Src)
A	AA	D	No Operation (NOP)
B	BB	DSno	Dest OR (NOT Src)
C	CC	S	Src (Source Copy)
D	DD	SDno	Src OR (NOT Dest)
E	EE	DSo	Dest OR Src
F	FF	1	Fill Dest with 1

ROP3 Codes (Listed in Hexadecimal and Reverse Polish Notation Pairs)

00	0	**40**	PSDnaa	**80**	DPSaa	**C0**	PSa
01	DPSoon	**41**	DPSxon	**81**	SPxDSxon	**C1**	SPDSnaoxn
02	DPSona	**42**	SDxPDxa	**82**	DPSxna	**C2**	SPDSonoxn
03	PSon	**43**	SPDSanaxn	**83**	SPDSnoaxn	**C3**	PSxn
04	SDPona	**44**	SDna	**84**	SDPxna	**C4**	SPDnoa
05	DPon	**45**	DPSnaon	**85**	PDSPnoaxn	**C5**	SPDSxoxn
06	PDSxnon	**46**	DSPDaox	**86**	DSPDSoaxx	**C6**	SDPnax
07	PDSaon	**47**	PSDPxaxn	**87**	PDSaxn	**C7**	PSDPoaxn
08	SDPnaa	**48**	SDPxa	**88**	DSa	**C8**	SDPoa
09	PDSxon	**49**	PDSPDaoxxn	**89**	SDPSnaoxn	**C9**	SPDoxn
0A	DPna	**4A**	DPSDoax	**8A**	DSPnoa	**CA**	DPSDxax
0B	PSDnaon	**4B**	PDSnox	**8B**	DSPDxoxn	**CB**	SPDSaoxn
0C	SPna	**4C**	SDPana	**8C**	SDPnoa	**CC**	S
0D	PDSnaon	**4D**	SSPxDSxoxn	**8D**	SDPSxoxn	**CD**	SDPono
0E	PDSonon	**4E**	PDSPxox	**8E**	SSDxPDxax	**CE**	SDPnao
0F	Pn	**4F**	PDSnoan	**8F**	PDSanan	**CF**	SPno
10	PDSona	**50**	PDna	**90**	PDSxna	**D0**	PSDnoa
11	DSon	**51**	DSPnaon	**91**	SDPSnoaxn	**D1**	PSDPxoxn
12	SDPxnon	**52**	DPSDaox	**92**	DPSDPoaxx	**D2**	PDSnax
13	SDPaon	**53**	SPDSxaxn	**93**	SPDaxn	**D3**	SPDSoaxn

ROP3 Codes (Listed in Hexadecimal and Reverse Polish Notation Pairs) (continued)

14	DPSxnon	**54**	DPSonon	**94**	PSDPSoaxx	**D4**	SSPxPDxax
15	DPSaon	**55**	Dn	**95**	DPSaxn	**D5**	DPSanan
16	PSDPSanaxx	**56**	DPSox	**96**	DPSxx	**D6**	PSDPSaoxx
17	SSPxDSxaxn	**57**	DPSoan	**97**	PSDPSonoxx	**D7**	DPSxan
18	SPxPDxa	**58**	PDSPoax	**98**	SDPSonoxn	**D8**	PDSPxax
19	SDPSanaxn	**59**	DPSnox	**99**	DSxn	**D9**	SDPSaoxn
1A	PDSPaox	**5A**	DPx	**9A**	DPSnax	**DA**	DPSDanax
1B	SDPSxaxn	**5B**	DPSDonox	**9B**	SDPSoaxn	**DB**	SPxDSxan
1C	PSDPaox	**5C**	DPSDxox	**9C**	SPDnax	**DC**	SPDnao
1D	DSPDxaxn	**5D**	DPSnoan	**9D**	DSPDoaxn	**DD**	SDno
1E	PDSox	**5E**	DPSDnaox	**9E**	DSPDSaoxx	**DE**	SDPxo
1F	PDSoan	**5F**	DPan	**9F**	PDSxan	**DF**	SDPano
20	DPSnaa	**60**	PDSxa	**A0**	DPa	**E0**	PDSoa
21	SDPxon	**61**	DSPDSaoxxn	**A1**	PDSPnaoxn	**E1**	PDSoxn
22	DSna	**62**	DSPDoax	**A2**	DPSnoa	**E2**	DSPDxax
23	SPDnaon	**63**	SDPnox	**A3**	DPSDxoxn	**E3**	PSDPaoxn
24	SPxDSxa	**64**	SDPSoax	**A4**	PDSPonoxn	**E4**	SDPSxax
25	PDSPanaxn	**65**	DSPnox	**A5**	PDxn	**E5**	PDSPaoxn
26	SDPSaox	**66**	DSx	**A6**	DSPnax	**E6**	SDPSanax
27	SDPSxnox	**67**	SDPSonox	**A7**	PDSPoaxn	**E7**	SPxPDxan
28	DPSxa	**68**	DSPDSonoxxn	**A8**	DPSoa	**E8**	SSPxDSxax
29	PSDPSaoxxn	**69**	PDSxxn	**A9**	DPSoxn	**E9**	DSPDSanaxxn
2A	DPSana	**6A**	DPSax	**AA**	D	**EA**	DPSao
2B	SSPxPDxaxn	**6B**	PSDPSoaxxn	**AB**	DPSono	**EB**	DPSxno
2C	SPDSoax	**6C**	SDPax	**AC**	SPDSxax	**EC**	SDPao
2D	PSDnox	**6D**	PDSPDoaxxn	**AD**	DPSDaoxn	**ED**	SDPxno
2E	PSDPxox	**6E**	SDPSnoax	**AE**	DSPnao	**EE**	DSo
2F	PSDnoan	**6F**	PDSxnan	**AF**	DPno	**EF**	SDPnoo
30	PSna	**70**	PDSana	**B0**	PDSnoa	**F0**	P
31	SDPnaon	**71**	SSDxPDxaxn	**B1**	PDSPxoxn	**F1**	PDSono
32	SDPSoox	**72**	SDPSxox	**B2**	SSPxDSxox	**F2**	PDSnao
33	Sn	**73**	SDPnoan	**B3**	SDPanan	**F3**	PSno

ROP3 Codes (Listed in Hexadecimal and Reverse Polish Notation Pairs) (continued)

34	SPDSaox	**74**	DSPDxox	**B4**	PSDnax	**F4**	PSDnao
35	SPDSxnox	**75**	DSPnoan	**B5**	DPSDoaxn	**F5**	PDno
36	SDPox	**76**	SDPSnaox	**B6**	DPSDPaoxx	**F6**	PDSxo
37	SDPoan	**77**	DSan	**B7**	SDPxan	**F7**	PDSano
38	PSDPoax	**78**	PDSax	**B8**	PSDPxax	**F8**	PDSao
39	SPDnox	**79**	DSPDSoaxxn	**B9**	DSPDaoxn	**F9**	PDSxno
3A	SPDSxox	**7A**	DPSDnoax	**BA**	DPSnao	**FA**	DPo
3B	SPDnoan	**7B**	SDPxnan	**BB**	DSno	**FB**	DPSnoo
3C	PSx	**7C**	SPDSnoax	**BC**	SPDSanax	**FC**	PSo
3D	SPDSonox	**7D**	DPSxnan	**BD**	SDxPDxan	**FD**	PSDnoo
3E	SPDSnaox	**7E**	SPxDSxo	**BE**	DPSxo	**FE**	DPSoo
3F	PSan	**7F**	DPSaan	**BF**	DPSano	**FF**	1

CHAPTER

8

Video API (GFVxAPI)

Overview

The GFVxAPI is an abstraction layer for both the Video Scalar (StretchBlt) and the Video Input (VI) module. The Video Scalar provides color space conversion (CSC) and smooth scaling of video. The VI module, which connects to the Video Input Port (VIP), accepts data from the VIP or the host CPU through a FIFO mechanism. Video data, either decimated or not, can be sent from the VI to a JPEG or MPEG encoder, and at the same time can go to memory through CSC for previewing, again either decimated or not. If a decimator is used in both the preview and the encoder paths, it has to have same decimation factor. There are also options to crop the data and to mirror and flip it.

The Video Scalar and the VI module accept YUV (4:2:2 or 4:2:0) and RGB (565) data as input. YUV data can be converted to YUV planar data or RGB data (RGB:565 for all processors, and RGB:888 or ARGB:8888 for GoForce 3D 4800). RGB data can be scaled, but cannot be converted to YUV, for all processors up to the GoForce 3D 4800.

Video Sources

The video source for the VI module can be the host CPU or a camera connected to the VIP. The video source for the Video Scalar can be the video memory surface or the output of the JPEG or MPEG decoder through a circular buffer. The GFVxAPI functions that deal *only* with the VIP have **`VIP`** in their function names.

Video Camera Interface

Usually a video camera connected through the VIP generates YUV 4:2:2 data (CCIR656 format) which can be encoded by the NVIDIA GPU. Some cameras generate JPEG-encoded images directly and the GFSDK can be configured to accept those JPEG-encoded bit streams.

Newer NVIDIA GPUs support the following camera interfaces that receive variable sized data:

- **Type A interface**: `VHSync` and `VVSync` length can be variable, and `VVClk` should be kept running.
- **Type B interface**: `VHSync` and `VVSync` length can be variable, and `VVClk` can be stopped during the image transfer period.
- **Type C interface**: `VHSync` and `VVSync` length are fixed, and `VVClk` should be kept running. The camera can insert `0xff` in the image as padding.

Video Coordinate Systems (Regular and Rotated)

A destination coordinate is always the same as a screen coordinate. The source coordinate stays the same, even if there is rotation.

GFVxAPI Reference

The GFVxAPI consists of the GFVxAPI functions (see "GFVxAPI Functions" on page 221) and the GFVxAPI data structures (see "GFVxAPI Data Structures" on page 233).

GFVxAPI Functions

This section describes the following GFVxAPI functions:

GFVxGetProperty()

This function returns information about the GFVxAPI, including the following:

- GFVxAPI module version
- VIP support
- Overlay support
- MPEG decoding and encoding
- JPEG decoding and encoding

It is a good practice to use this function to query for the API version and its capabilities before using the rest of the GFVxAPI functions.

Function Prototype

```
GF_RETTYPE  GFVxGetProperty(
    GF_HANDLE    VXhandle,
    PGFPROPERTY  pVXProp );
```

Parameters

VXhandle	Handle specific to the GFVxAPI. Obtained by calling "GFRmComponentGet()" on page 26.
pVXProp	Pointer to GFVxAPI properties. See "GFPROPERTY" on page 233.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxBlt()

GFVxBlt() copies the **pSrcRect** area in **ppSrcSurf** to the **pDestRect** area in **pDestSurf**. If **pSrcRect** differs in size from **pDestRect**, the function scales the source data to fit the destination rectangle before copying it to the destination. If the source and destination surfaces have different color formats, **GFVxBlt()** provides limited color format conversions.

Color format conversions from a YUV420 source are supported for these destination formats:

- YUV420
- RGB565
- ARGB8888 (GoForce 3D 4800)
- RGB888 (GoForce 3D 4800)

Color format conversions from a YUV422 source are supported for these destination formats:

- YUV420
- YUV422
- RGB565
- ARGB8888 (GoForce 3D 4800)

- RGB888 (GoForce 3D 4800)

Function Prototype

```
GF_RETTYPE  GFVxBlt(
    GF_HANDLE  VXhandle,
    PGFVXBLT   pBlt );
```

Parameters

VXhandle	Handle specific to the GFVxAPI. Obtained by calling "GFRmComponentGet()" on page 26.
pBlt	Pointer to GFVXBLT structure. See "GFVXBLT" on page 239.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Interaction with GFVXBLT.BltOption. If **GF_VX_AUTO_BLT** is on, **ppDestSurf** points to a destination array of surfaces. The number of elements in the array of destination surfaces is specified by **numofDestSurf**. Pointer **ppSrcSurf** points to a source array of surfaces. The total number of elements in the array of source surfaces is specified by **numofSrcSurf**. The GFVxAPI only prepares to do the automatic blit, it does not really do it. It is certain functions, such as **GFVxVIPUpdate()** and **GFMxDecSetMBs()**, that automatically trigger the blit. The first time, the GFVxAPI copies the first surface in **ppSrcSurf** to the first surface in **ppDestSurf**. The second time, the GFVxAPI copies the second surface in the **ppSrcSurf** to the second surface in the **ppDestSurf** array. (If there is only one surface in the **ppSrcSurf** array, the GFVxAPI always uses that one. If there is only one surface in the **ppDestSurf** array, the GFVxAPI always copies to that one.) The GFVxAPI always does the auto blit in this order.

If **GF_VX_AUTO_BLT** is off, this function always blits from the first surface in **ppSrcSurf** to the first surface in **ppDestSurf**.

If **GF_VX_AUTO_DISPLAY** is on, **GFVxBlt()** automatically shows the destination surface on the screen. the application does not need call **GFVxFlip()** to show the surface.

If **GF_VX_BLT_SRC_KEY** is on, **GFVxBlt()** compares the **colorKey** with source surface. If the colors match, **GFVxBlt()** does not copy this pixel to destination surface.

If **GF_VX_BLT_DEST_KEY** is on, **GFVxBlt()** compares the **colorKey** with destination surface. If the color match, **GFVxBlt** overwrites this pixel with the corresponding pixel from source surface.

GFVxFlip()

This function displays **pSurf**. If **pSurf** is an overlay surface, this function displays the **pSurf** to **pDestRect** area that has been set by **GFVxUpdateOverlay()**. If **pSurf** is not an overlay surface, this function displays all of **pSurf**.

Normally, **GFVxFlip()** flips either overlay surface 1 or overlay surface 2 on the primary surface. If there is only one overlay surface, **GFVxFlip()** should not been called. **GFVxFlip()** can show either the primary surface or an off-screen surface on the display screen. If there is only one primary surface, **GFVxFlip()** should not been called.

Function Prototype

```
GF_RETTYPE  GFVxFlip(
    GF_HANDLE  VXhandle,
    PGFVXFLIP  pFlip );
```

Parameters

VXhandle	Handle specific to the GFVxAPI.
pFlip	Pointer to PGFVXFLIP structure. See "GFVXFLIP" on page 240.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxUpdateOverlay()

The application needs to check properties (see "GFVxGetProperty()" on page 221) to see if the current NVIDIA media processor supports overlays. If it does, this function is supported.

This function sets the destination and source rectangle areas for overlay support. It also sets the destination color key. The media processor checks the destination rectangle area, which is set by **pDestRect** in **GFVxSetVideo()**. If **GFVX_UO_DEST_KEY** is set, if alpha blending is enabled, and if the color in

this area matches **`BlendingColorKey`**, the following equation is used to generate the final pixel:

```
src * AlphaValue + dest * (1-AlphaValue)
```

Otherwise, the original pixel is not updated.

The application should paint that area using the color key color. The application does not need repaint the area every frame, only when it is necessary. For example, in the CE environment if the drop down menu overwrites this area, the color will not match the color key. The media processor does not let the video image appear on top of the menu. But when the menu is closed, the application gets the repaint message and should repaint this area using the color key so that the media processor will put the video on top of it.

The application should choose an exotic color as the color key to avoid conflicts with the colors in the menu. If **`GFVX_UO_SRC_KEY`** is set the media processor checks the source image, and if the color matches the color key the destination pixel is shown. If they do not match or if no flag is set, the source image is shown.

Function Prototype

```
GF_RETTYPE  GFVxUpdateOverlay(
    GF_HANDLE           VXhandle,
    PGFVXUPDATEOVERLAY  pOverlay );
```

Parameters

`VXhandle`	Handle specific to the GFVxAPI.
`pOverlay`	Pointer to `GFVXDECUPDATEOVERLAY` structure. See "GFVXUPDATEOVERLAY" on page 241.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFVxVIPSetVIP()

This function sets up the VIP.

Function Prototype

```
GF_RETTYPE  GFVxVIPSetVIP(
    GF_HANDLE     VXhandle,
    PGFVXVIPINFO  pVIPInfo );
```

Parameters

VXhandle	Handle specific to the GFVxAPI.
pVIPINFO	Pointer to GFVXVIPINFO structure. See "GFVXVIPINFO" on page 236.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxVIPGetProperty()

This function returns a variety of information about the VIP module.

Function Prototype

```
GF_RETTYPE  GFVxVIPGetProperty(
    GF_HANDLE         VXhandle,
    PGFVXVIPPROPERTY  pVIPProp );
```

Parameters

VXhandle	Handle specific to the GFVxAPI.
pVIPProp	Pointer to VIP properties. See "GFVXVIPPROPERTY" on page 234.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxVIPUpdate()

This function starts or stops the display of the video image from the VIP.

Function Prototype

```
GF_RETTYPE  GFVxVIPUpdate(
    GF_HANDLE       VXhandle,
    PGFVXVIPUPDATE  pUpdate );
```

Parameters

VXhandle	Handle specific to the GFVxAPI.
pUpdate	Pointer to GFVXVIPUPDATE. See "GFVXVIPUPDATE" on page 243.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxVIPFeedImage()

GFVxVIPFeedImage() feeds the source image to the VIP. It only supports YUV420, YUV422 (YUYV, TVYU, UYVY, VYUY), and YUV422 planar formats.

Function Prototype

```
GF_RETTYPE  GFVxVIPFeedImage(
    GF_HANDLE          VXhandle,
    PGFVXVIPFEEDIMAGE  pImage );
```

Parameters

VXhandle	Handle specific to the GFVxAPI.
pImage	Pointer to GFVXVIPFeedImage. See "GFVXVIPFEEDIMAGE" on page 245.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxSleep()

This function puts the video unit into sleep mode. This is not implemented by default, but is intended to be implemented only in GFSDK ports to platforms that need this feature.

Function Prototype

```
GF_RETTYPE GFVxSleep(GF_HANDLE VxHandle);
```

Parameters

VxHandle	Handle to VxAPI

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxWakeup()

This function wakes up the video unit from sleep mode. This is not implemented by default, but is intended to be implemented only in GFSDK ports to platforms that need this feature.

Function Prototype

```
GF_RETTYPE GFVxWakeup(GF_HANDLE VxHandle);
```

Parameters

VXHandle	Handle to VxAPI

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxVIPGPIO()

This function manipulates the GPIO pin signals in the video input port. The supported operations are clearing to `0`, setting to `1`, and getting the status of the input enable, output enable, and data bits.

Function Prototype

```
GF_RETTYPE  GFVxVIPGPIO(
    GF_HANDLE           VXhandle,
    GFVX_VIP_GPIO_TYPE  gpio,
    NvU32               operation,
    PGFGPIOSTATUS       pGPIOStatus );
```

Parameters

`VXhandle`	Handle specific to the GFVxAPI.
`gpio`	One of the supported VIP GPIOs. See "GFVX_VIP_GPIO_TYPE Enumeration Type" below.
`operation`	Combinations of different operations. See "GFGPIOSTATUS" on page 16.
`pGPIOStatus`	Pointer to the returned status information.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFVX_VIP_GPIO_TYPE Enumeration Type

```
typedef enum{
    GFVX_VIP_VGP0 = 1,
    GFVX_VIP_VGP1,
    GFVX_VIP_VGP2,
    GFVX_VIP_VGP3,
    GFVX_VIP_VGP4,
    GFVX_VIP_VGP_VID0,
    GFVX_VIP_VGP_VID1,
    GFVX_VIP_VGP_VID2,
    GFVX_VIP_VGP_VID3,
    GFVX_VIP_VGP_VID4,
    GFVX_VIP_VGP_VID5,
    GFVX_VIP_VGP_VID6,
    GFVX_VIP_VGP_VID7,
    GFVX_VIP_VGP_VCLK,
    GFVX_VIP_VGP_VHSYNC,
    GFVX_VIP_VGP_VVSYNC
} GFVX_VIP_GPIO_TYPE;
```

GFVxInterruptControl()

This function provides component-level interrupt control for the GFVxAPI.

Function Prototype

```
GF_RETTYPE GFVxInterruptControl(
    GF_HANDLE                       VxHandle,
    GFVX_INTERRUPT_TYPE             IntType,
    GFVX_INTERRUPT_OPERATION_TYPE   op,
    void                            *pData );
```

Parameters

VxHandle	Handle specific to the GFVxAPI
IntType	GFVxAPI component-related interrupt type as defined in “GFVX_INTERRUPT_TYPE” on page 246

Parameters (continued)

op	Interrupt operation as defined in "GFVX_INTERRUPT_OPERATION_TYPE" on page 245.
pData	Pointer to data that is passed in or out by this function per interrupt operation. • It is defined as the interrupt status when op is GFVX_INTERRUPT_QUERY_STATUS. • It is defined as the YFIFO threshold values when op is GFVX_INTERRUPT_SET_Y_FIFO_THRESHOLD or GFVX_INTERRUPT_SET_V_COUNTER_THRESHOLD. • It is defined as the vertical counter value when op is GFVX_INTERRUPT_SET_V_COUNTER_THRESHOLD or GFVX_INTERRUPT_GET_V_COUNTER_THRESHOLD.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.
GF_ERROR_NO_SUPPORT	If the interrupt operation is not supported.

GFVxGetAttribute()

This function returns GFVxAPI attributes as defined in "GFVXATTRIBUTES" on page 247.

Function Prototype

```
GF_RETTYPE GFVxGetAttribute(
    GF_HANDLE  VxHandle,
    NvU32      aid,
    NvU32      *attr );
```

Parameters

VxHandle	Handle specific to the GFVxAPI.
aid	It is a value defined in GFVXATTRIBUTES.
attr	Pointer to the attribute.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxSetAttribute()

This function sets the GFVxAPI attributes as defined in "GFVXATTRIBUTES" on page 247,

Function Prototype

```
GF_RETTYPE GFVxSetAttribute(
    GF_HANDLE VxHandle,
    NvU32     aid,
    NvU32     attr);
```

Parameters

VxHandle	Handle to VxAPI
aid	Attribute type identifier define in GFVXATTRIBUTES.
attr	Attribute value based on the attribute type.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFVxAPI Data Structures

This section describes the following GFVxAPI data structures:

GFPROPERTY

This structure is described in "GFPROPERTY" on page 10. The GFVxAPI **`GFPROPERTY`** capabilities are defined below.

GFPROPERTY Definitions

```
/*
    Capability Definitions
*/
#define GFVX_CAP_CSC                            0x00000001
    // Color space conversion available.
#define GFVX_CAP_VIP                            0x00000002
    // VIP functions available.
#define GFVX_CAP_ENLARGE                        0x00000004
    // Enlarge source image.
#define GFVX_CAP_SHRINK                         0x00000008
    // Shrink source image.
#define GFVX_CAP_COLOR_CONTROL                  0x00000010
    // Support color control.
#define GFVX_CAP_OVERLAY                        0x00000020
    // Support overlay.
#define GFVX_CAP_MPEGDEC                        0x00000040
    // Support MPEG-4 decoder.
```

GFPROPERTY Definitions (continued)

```
#define GFVX_CAP_MPEGENC                        0x00000080
    // Support MPEG-4 encoder.
#define GFVX_CAP_JPEGDEC                        0x00000100
    // Support JPEG decoder.
#define GFVX_CAP_JPEGENC                        0x00000200
    // Support JPEG encoder.
/*
    The following capabilities are for the GoForce 3D 4800.
*/
#define GFVX_CAP_ALPHA_BLEND                    0x00000400
    // Support alpha blending.
#define GFVX_CAP_EXTEND_RGB                     0x00000800
    // Support extend RGB format: 24/32 bpp.
#define GFVX_CAP_ENCODE_TYPE                    0x00001000
    // Support encoded type input: TYPE_A/B/C.
```

GFVXVIPPROPERTY

Used by "GFVxVIPGetProperty()" on page 226.

GFVXVIPPROPERTY Structure

```
typedef struct  _GFVXVIPPROPERTY {
    NvU32  Flag;
    NvU32  DestFormat;
    NvU32  HMaxNumerator;
    NvU32  HMaxDenominator;
    NvU32  VMaxNumerator;
    NvU32  VMaxDenominator;
} GFVXVIPPROPERTY, *PGFVXVIPPROPERTY;
```

GFVXVIPPROPERTY Fields

`Flag`	See "GFVXVIPPROPERTY Definitions".
`DestFormat`	VIP output format. Bit-wise `OR` is used on YUV data formats, and the RGB data format is the highest data format supported for VIP output. If the format is `GF_SURFACE_ARGB8888`, `GF_SURFACE_RGB565` and `GF_SURFACE_RGB888` are also supported if the media processor supports a `GF_SURFACE_RGB888` surface.
`HMaxNumerator`	Horizontal maximum numerator. Application chooses a number from 1 to `HMaxNumerator`.

GFVXVIPPROPERTY Fields (continued)

`HMaxDenominator`	Horizontal maximum denominator. Application chooses a number from 1 to `HMaxDenominator`.
`VMaxNumerator`	Vertical maximum numerator. Application chooses a number from 1 to `VMaxNumerator`.
`VMaxDenominator`	Vertical maximum denominator. Application chooses a number from 1 to `VMaxDenominator`.

GFVXVIPPROPERTY Definitions

```
/*
    GFVXVIPPROPERTY Flag
*/
#define GFVX_VIP_CAP_SHRINK                   0x00000001
    // Shrink.
#define GFVX_VIP_CAP_ENLARGE                  0x00000002
    // Enlarge.
#define GFVX_VIP_CAP_PARALLEL                 0x00000004
    // Parallel input.
#define GFVX_VIP_CAP_SERIAL                   0x00000008
    // Serial input.
#define GFVX_VIP_CAP_RGB_VH_FLIP              0x00000010
    // Vertical and horizontal flip for output RGB image.
#define GFVX_VIP_CAP_OUTPUT_VCLOCK_VHSYNC     0x00000020
    // Supply VClock and VHSync from GF processor.
#define GFVX_VIP_CAP_OUTPUT_MASTER_CLOCK      0x00000040
    // Supply master clock to camera chip.
#define GFVX_VIP_CAP_TYPE_A                   0x00000080
    // VIP can accept JPEG image from Camera through
    // type A interface.
#define GFVX_VIP_CAP_TYPE_B                   0x00000100
    // VIP can accept JPEG image from Camera through
    // type B interface.
#define GFVX_VIP_CAP_TYPE_C                   0x00000200
    // VIP can accept JPEG image from Camera through
    // type C interface.
```

GFVXVIPINFO

Used by "GFVxVIPSetVIP()" on page 226.

GFVXVIPINFO Structure

```
typedef struct  _GFVXVIPINFO {
    NvU32  Flag;
    NvU32  HOffset;
    NvU32  HActive;
    NvU32  VOffset;
    NvU32  VActive;
    NvU32  ColorFormat;
    NvU32  SerialDelay;                    // Do not use
    NvU32  NewTiming;
    NvU32  HNumerator;
    NvU32  HDenominator;
    NvU32  VNumerator;
    NvU32  VDenominator;
    NvU32  MClockFreq;
    NvU32  VClockFreq;
    NvU32  VHPulseWidth;
    NvU32  VHPulsePeriod;
    NvU32  VVPulseWidth;
    NvU32  VVPulsePeriod;
    NvU32  VVDelay;
} GFVXVIPINFO, *PGFVXVIPINFO;
```

GFVXVIPINFO Fields

`Flag`	See `GFVXVIPINFO` definitions.
`HOffset`	VIP input signal: horizontal offset.
`HActive`	VIP input signal: horizontal active width.
`VOffset`	VIP input signal: vertical offset.
`VActive`	VIP input signal: vertical active height.
`ColorFormat`	See `GFRMSURFACE ColorFormat`.
`SerialDelay`	VIP serial input delay. (*Reserved for future.*)
`NewTiming`	See `GFVXVIPINFO` definitions.
`HNumerator`	Number of pixels to keep from a group of incoming pixels.
`HDenominator`	Group size.

GFVXVIPINFO Fields (continued)

VNumerator	Number of line to keep from a group of incoming lines.
VDenominator	Group size.
MClockFreq	Master clock frequency required for camera.
VClockFreq	Clock frequency required for VIP input signal.
The next fields must be filled if GFVX_VIP_INTERNAL_VHSYNC is set in Flag.	
VHPulseWidth	VHSYNC pulse width in terms of number of VCLK (1-8).
VHPulsePeriod	VHSYNC pulse period in terms of number of VCLK (32-2048).
VVPulseWidth	VVSYNC pulse width in term of number of VCLK (1-8).
VVPulsePeriod	VVSYNC pulse period in terms of number of VCLK (32-1024).
VVDelay	Specify VCLK cycles from leading edge of VHSYNC to leading edge of VVSYNC (-2 to 13).

GFVXVIPINFO Definitions

```
/*
    GFVXVIPINFO Flag
*/
#define GFVX_VIP_PARALLEL_INPUT             0x00000001UL
    // 8-bit/clock video data.
#define GFVX_VIP_SERIAL_INPUT               0x00000002UL
    // VHSYNC and VVSYNC must be in the stream.
#define GFVX_VIP_HVSYNC_IN_STREAM           0x00000004UL
    // VHSYNC and VVSYNC included in stream.
#define GFVX_VIP_EXTERNAL_VHSYNC            0x00000008UL
    // VHSYNC from external pin, not the stream.
#define GFVX_VIP_INTERNAL_VHSYNC            0x00000010UL
    // VHSYNC from NVIDIA processor, not the stream.
#define GFVX_VIP_EXTERNAL_VCLK              0x00000020UL
    // VCLK from external.
#define GFVX_VIP_INTERNAL_VCLK              0x00000040UL
    // VCLK from NVIDIA processor.
#define GFVX_VIP_DETECT_FIELD               0x00000080UL
    // Detect field 0 or field 1.
    // Does not work for internal VHSYNC.
#define GFVX_VIP_RGB_H_FLIP                 0x00000100UL
    // Horizontal flip.
```

GFVXVIPINFO Definitions (continued)

```
#define GFVX_VIP_RGB_V_FLIP                 0x00000200UL
    // Vertical flip.
#define GFVX_VIP_HOST_IMAGE                 0x00000400UL
    // CPU feeds image to VIP.
#define GFVX_VIP_CHANGE_DEFAULT_TIMING      0x00000800UL
    // Change default timing for VHSYNC and VCLK, such as
    // polarity, active edge, serial frame polarity, size,
    // field info (e.g. new NewTiming, coordinate, etc.).
#define GFVX_VIP_ASK_FOR_MCLOCK             0x00001000UL
    // Ask for Master Clock for camera. When set,
    // MClockFreq must be set to ideal
    // clock that camera wants. GFVxAPI provides
    // the best match of clock to camera.
#define GFVX_VIP_PRE_GENERATE_MCLOCK        0x80000000UL
    // Request pre-generation of Master Clock for camera.
    // Useful for scanning for right camera.
#define GFVX_VIP_TYPE_A                     0x00002000UL
    // Camera is running at type A interface.
#define GFVX_VIP_TYPE_B                     0x00004000UL
    // Camera is running at type B interface.
#define GFVX_VIP_TYPE_C                     0x00008000UL
    // Camera is running at type C interface.
#define GFVX_VIP_DEST_BYTE_SWAP             0x00010000UL
    // Enables destination byte swap.
#define GFVX_VIP_BYPASS_MCLOCK_PIN          0x20000000UL
    // Bypass ANY manipulation on internal Master
    // Clock generation output pin. Should be mutually
    // exclusive to GFVX_VIP_PRE_GENERATE_MCLCOK and/or
    // GFVX_VIP_ASK_FOR_MCLOCK.
#define GFVX_VIP_BYPASS_NON_MCLOCK_PINS     0x40000000UL
    // Bypass all manipulations on VCLK, VHSYNC,
    // VVSYNC, and VID[7:0] pins.
#define GFVX_VIP_PRE_GENERATE_MCLOCK        0x80000000UL
    // Request pre-generation of Master Clock
    // for Camera. Useful for scanning for right camera.
/*
    GFVXVIPINFO NewTiming
*/
#define GFVX_VIP_PDL_FALLING_VCLK           0x00000001UL
    // Parallel data latched at falling edge of VCLK.
    // Default is rising edge.
```

GFVXVIPINFO Definitions (continued)

```
#define GFVX_VIP_SDL_RISING_VCLK            0x00000001UL
    // Serial data latched at rising edge of VCLK.
    // Default is falling edge.
#define GFVX_VIP_VHSYNC_ACTIVE_LOW          0x00000002UL
    // VHSYNC active low. Default is active high.
#define GFVX_VIP_VVSYNC_ACTIVE_LOW          0x00000004UL
    // VVSYNC active low. Default is active high.
#define GFVX_VIP_VHSYNC_ACTIVE_EDGE_LEAD    0x00000008UL
    // VHSYNC active leading edge. Default is trailing edge.
#define GFVX_VIP_VVSYNC_ACTIVE_EDGE_LEAD    0x00000010UL
    // VVSYNC active leading edge. Default is trailing edge.
#define GFVX_VIP_SFRAME_ACTIVE_LOW          0x00000020UL
    // Serial video frame sync active low,
    // falling edge indicates start of data frame.
    // Default is active high, rising
    // edge indicates start of data frame.
#define GFVX_VIP_SFRAME_SIZE_16             0x00000040UL
    // Serial video input is 16 bits per frame.
    // Default is 8 bit per frame.
#define GFVX_VIP_FILED_0_BOTTOM             0x00000080UL
    // Field 0 is bottom/even field.
    // Default is top/odd field.
#define GFVX_VIP_VCLK_OUTPUT_ACTIVE_LOW     0x00000100UL
    // VCLK output from chip active low.
    // Default is active high.
```

GFVXBLT

Used by "GFVxBlt()" on page 222.

GFVXBLT Structure

```
typedef struct  _GFVXBLT {
    PGFRMSURFACE   *ppDestSurf;
    PGFRMSURFACE   *ppSrcSurf;
    PGFRECT        pDestRect;
    PGFRECT        pSrcRect;
    NvU32          numofDestSurf;
    NvU32          numofSrcSurf;
    NvU32          colorKey;
    NvU32          BltOption;
} GFVXBLT, *PGFVXBLT;
```

GFVXBLT Fields

`*ppDestSurf`	Pointer to an array of destination surfaces or to an array of destination surfaces that accommodate automatic blit.
`*ppSrcSurf`	Pointer to an array of source surfaces or to an array of source surfaces that accommodate automatic blit.
`pDestRect`	Destination rectangle. See "GFRECT" on page 11.
`pSrcRect`	Source rectangle.
`NumofDestSurf`	If `GFVX_AUTO_BLT` flag is set, this field must be filled.
`NumofSrcSurf`	If `GFVX_AUTO_BLT` flag is set, this field must be filled.
`colorKey`	Color key value. Format is RGB888, 24 bpp.
`BltOption`	`GFVX_AUTO_BLT` or `GFVX_AUTO_DISPLAY`.

`GFVXBLT` Definitions

```
#define GFVX_BLT_DISABLE_WAIT_VSYNC           0x00000080
    // Disable VSync wait.
    // Disable GFVxBlt() synchronization with VSync.
#define GFVX_BLT_SRC_KEY                      0x00000100
    // Source color key option is not very useful when
    // scaling is used. Input of source data is compared and
    // it is copied when color key doesn't match.
#define GFVX_AUTO_BLT                         0x00000400
    // AutoBlt can be used to fire StretchBlt automatically
    // at the end of the video input frame from the video
    // input port (camera). This requires double buffering
    // of the input and output of StretchBlt.
#define GFVX_AUTO_DISPLAY                     0x00000800
    // In the autodisplay option, the VI module triggers
    // StretchBlt() which in turn sends the buffer address to
    // the display controller (GC). VI input goes into
    // double buffers which become the StretchBlt source and
    // the output of the StretchBlt is also double buffered.
```

GFVXFLIP

Used by "GFVxFlip()" on page 224.

GFVXFLIP Structure

```
typedef struct  _GFVXFLIP {
    PGFVXSURF  pBackGroundSurf;
    PGFVXSURF  pForeGroundSurf;
```

GFVXFLIP Structure (continued)

```
    NvU32       FlipOption;
} GFVXFLIP, *PGFVXFLIP;
```

GFVXFLIP Fields

pBackGroundSurf	Pointer to the surface to be flipped to the background. This surface will not be seen after the GFVxFlip() call.
pForeGroundSurf	Pointer to the surface to be flipped to the foreground. This surface will be seen after the GFVxFlip() call.
FlipOption	Not used for current version; must be set to 0.

GFVXUPDATEOVERLAY

Used by "GFVxUpdateOverlay()" on page 224.

GFVXUPDATEOVERLAY Structure

```
typedef struct  _GFVXUPDATEOVERLAY {
    PGFRMSURFACE  pDestSurf;
    PGFRMSURFACE  pSrcSurf;
    PGFRECT       pDestRect;
    PGFRECT       pSrcRect;
    NvU32         numofSrcSurf;
    NvU32         ColorKey;
    NvU32         UpdateOption;
    NvU32         BlendingColorKey;
    NvU32         AlphaValue;
} GFVXUPDATEOVERLAY;
```

GFVXUPDATEOVERLAY Fields

pDestSurf	Destination surface pointer
pSrcSurf	Source surface pointer. If this pointer is NULL, disable overlay.
PdestRect	Destination rectangle. See "GFRECT" on page 11.
pSrcRect	Source rectangle.
numofSrcSurf	Number of source surfaces.
ColorKey	Color key value in ARGB32 format. ColorKey takes effect only when GFVX_UO_SRC_COLOR_KEY or GFVX_UO_DEST_COLOR_KEY is set.

GFVXUPDATEOVERLAY Fields (continued)

`UpdateOption`	Rotation options and color keying option. To do rotation on an overlay surface, specify `GF_SURFACE_ROTATE_xxx` in the `UpdateOption` field.
`BlendingColorKey`	Blending color key value in RGB32 format. Value takes effect only when `GFVX_UO_SRC_COLOR_KEY` or `GFVX_UO_DEST_COLOR_KEY` is set.
`AlphaValue`	This alpha value is used to do blending when the blending color key is matched. Each color channel has is own alpha value. The low 8 bits [7:0] are for blue, [15:8] are for green, and [23:16] are for red.

GFVXUPDATEOVERLAY Definitions

```
#define GFVX_UO_ROTATE_0              GF_SURFACE_ROTATE_0
    // Normal mode.
#define GFVX_UO_ROTATE_90            GF_SURFACE_ROTATE_90
    // Rotate 90 degrees.
#define GFVX_UO_ROTATE_180          GF_SURFACE_ROTATE_180
    // Rotate 180 degrees.
#define GFVX_UO_ROTATE_270          GF_SURFACE_ROTATE_270
    // Rotate 270 degrees.
#define GFVX_UO_H_FLIP                 GF_SURFACE_H_FLIP
#define GFVX_UO_V_FLIP                 GF_SURFACE_V_FLIP

#define GFVX_UO_SRC_COLOR_KEY                 0x00010000
    // Source color key is not useful for video overlay
    // because changing video is the source.
#define GFVX_UO_DEST_COLOR_KEY                0x00020000
    // Destination color key option is used to put a video
    // pixel from the overlay window wherever a graphics
    // pixel matches the color key.
#define GFVX_UO_ALPHA_BLENDING                0x00040000
    // Enable alpha blending. Alpha is from AlphaValue.
#define GFVX_UO_COLOR_KEY_ALPHA_ONLY          0x00080000
    // ColorKey has alpha value. GFVxUpdateOverlay()
    // compares this alpha value with surface alpha only and
    // ignores the RGB value.
```

GFVXUPDATEOVERLAY Definitions (continued)

```
#define GFVX_UO_COLOR_KEY_ALPHA_COLOR         0x00100000
    // ColorKey has alpha value and color value.
    // GFVxUpdateOverlay() compares this alpha value and
    // color value with surface alpha and color value.
    // If GFVX_UO_COLOR_KEY_ALPHA_ONLY and
    // GFVX_UO_COLOR_KEY_ALPHA_COLOR are not set,
    // GFVxUpdateOverlay() assumes RGB portion is valid in
    // ColorKey and compares the RGB value with surface's
    // RGB value.
#define GFVX_UO_BLD_COLOR_KEY_ALPHA_ONLY      0x00200000
    // BlendingColorKey has alpha value.
    // GFVxUpdateOverlay() compares this alpha value with
    // surface alpha only and ignores the RGB value.
#define GFVX_UO_BLD_COLOR_KEY_ALPHA_COLOR     0x00400000
    // BlendingColorKey has alpha value and color value.
    // GFVxUpdateOverlay() compares this alpha value and
    // color value with surface alpha and color value.
    // If GFVX_UO_BLD_COLOR_KEY_ALPHA_ONLY and
    // GFVX_UO_BLD_COLOR_KEY_ALPHA_COLOR are not set,
    // GFVxUpdateOverlay() assumes RGB portion is valid in
    // ColorKey and compares the RGB value with surface's
    // RGB value.
```

GFVXVIPUPDATE

Used by "GFVxVIPUpdate()" on page 227.

If the `*ppSurf` field is NULL and the `VIP_START` flag is on, the GFVxAPI still turns on the VIP. Data from the VIP is not displayed; it just flows to the MPEG or JPEG encoder pre-processor if an encoder is enabled. This mode is useful for the following two cases:

- There is not enough memory for JPEG encoding. Application could turn off the screen, enable this mode, and let the JPEG encoder utilize screen memory (the primary surface).
- In single-shot mode, the application does not want to display the video image on the screen after enabling the JPEG encoder, allowing faster captures and power savings, and freeing bandwidth. After the image is captured, the application enables the JPEG decoder to decode the captured image and show it on the screen.

If *ppSurf is not NULL, data from the VIP is copied to the first output surface, and video input (VI) color space conversion (CSC) is provided if the output surface is RGB format.

The output surface for VIP data can have following formats:

- GF_SURFACE_RGB565. 16 bits per pixel. VI CSC is used.
- GF_SURFACE_RGB888. 24 bits per pixel packed. VI CSC is used. It is only available for the GoForce 3D 4800 and later.
- GF_SURFACE_ARGB8888. 32 bits per pixel. VI CSC is used. It is only available for the GoForce 3D 4800.
- GF_SURFACE_YUV422. 32 bits per two pixels. No CSC is provided. Data from the VIP is stored "as is" in a memory surface.

Any other format is considered to be YUV422, and the data is stored "as is."

GFVXVIPUPDATE Structure

```
typedef struct  _GFVXVIPUPDATE {
    PGFRMSURFACE  *ppSurf;
    NvU32         numofSurf;
    NvU16         XStart;
    NvU16         YStart;
    NvU32         UpdateOption;
} GFVXVIPUPDATE;
```

GFVXVIPUPDATE Fields

ppSurf	Pointer to an array of VIP output surfaces.
numofSurf	If VIP_AUTO_FLIP is on, this field tells how many surfaces are in the ppSurf array.
XStart	X position for the output from the VIP.
YStart	Y position for the output from the VIP.
UpdateOption	See "GFVXVIPUPDATE Definitions".

GFVXVIPUPDATE Definitions

```
/*
    UpdateOption Defines
*/
#define GFVX_VIP_START                          0x00000001
    // Start VIP.
#define GFVX_VIP_STOP                           0x00000002
    // Stop VIP.
```

GFVXVIPUPDATE Definitions (continued)

```
#define GFVX_VIP_AUTO_FLIP   (0x00000004|GFVX_VIP_START)
    // Auto flip to display the VIP output surfaces.
#define GFVX_VIP_AUTO_TRIGER_BLT (0x00000008|GFVX_VIP_START)
    // Must set GFVX_AUTO_BLT option in GFVxBlt.
```

GFVXVIPFEEDIMAGE

Used by "GFVxVIPFeedImage()" on page 227.

GFVXVIPFEEDIMAGE Structure

```
typedef struct  _GFVXVIPFEEDIMAGE {
    PGFRMSURFACE  pSurf;
    PGFRECT       pSrcRect;
} GFVXVIPFEEDIMAGE;
```

GFVXVIPFEEDIMAGE Fields

`pSurf`	Surface that contains the image. `GFVxVIPFeedImage()` only supports YUV420, YUV422 (YUYV, TVYU, UYVY, VYUY), and YUV422 planar formats. `pSurf` must be one of those color formats.
`pSrcRect`	Rectangular area that is to be fed in. See "GFRECT" on page 11.

GFVX_INTERRUPT_OPERATION_TYPE

This data type is used by "GFVxInterruptControl()" on page 230.

GFVX_INTERRUPT_OPERATION_TYPE Enumeration Type

```
typedef enum {
    GFVX_INTERRUPT_ENABLE,
    GFVX_INTERRUPT_DISABLE,
    GFVX_INTERRUPT_CLEAR,
    GFVX_INTERRUPT_QUERY_STATUS,
    GFVX_INTERRUPT_SET_Y_FIFO_THRESHOLD,
    GFVX_INTERRUPT_GET_Y_FIFO_THRESHOLD,
    GFVX_INTERRUPT_SET_V_COUNTER_THRESHOLD,
    GFVX_INTERRUPT_GET_V_COUNTER_THRESHOLD
} GFVX_INTERRUPT_OPERATION_TYPE;
```

GFVX_INTERRUPT_TYPE

This data type is used by "GFVxInterruptControl()" on page 230.

GFVX_INTERRUPT_TYPE Enumeration Type

```
typedef enum {
    GFVX_VID0PIN_RISING_EDGE_INTR =                0x1,
        // VID[0] pin rising edge interrupt.
    GFVX_VID1PIN_RISING_EDGE_INTR =                0x2,
        // VID[1] pin rising edge interrupt.
    GFVX_VID2PIN_RISING_EDGE_INTR =                0x4,
        // VID[2] pin rising edge interrupt.
    GFVX_VID3PIN_RISING_EDGE_INTR =                0x8,
        // VID[3] pin rising edge interrupt.
    GFVX_VID4PIN_RISING_EDGE_INTR =               0x10,
        // VID[4] pin rising edge interrupt.
    GFVX_VID5PIN_RISING_EDGE_INTR =               0x20,
        // VID[5] pin rising edge interrupt.
    GFVX_VID6PIN_RISING_EDGE_INTR =               0x40,
        // VID[6] pin rising edge interrupt.
    GFVX_VID7PIN_RISING_EDGE_INTR =               0x80,
        // VID[7] pin rising edge interrupt.

    GFVX_VERTICAL_COUNTER_THRESHOLD_INTR =       0x100,
        // Vertical counter threshold interrupt.
    GFVX_VHSYNC_RISING_INTR =                    0x200,
        // VHSYNC pin rising edge interrupt.
    GFVX_VVSYNC_RISING_INTR =                    0x400,
        // VVSYNC pin rising edge interrupt.

    GFVX_VIDEO_IN_FIELD_DATA_RECV_INTR =         0x800,
        //Video-in field data received interrupt enable.

    GFVX_EARLY_VIDEO_INTR =                     0x1000,
        // Early video-in field data received interrupt.
    GFVX_Y_FIFO_THRESHOLD_INTR =                0x2000,
        // Y-FIFO threshold interrupt.

    GFVX_VID0PIN_FALLING_EDGE_INTR =           0x10001,
        // VID[0] pin falling edge interrupt.
    GFVX_VID1PIN_FALLING_EDGE_INTR =           0x20002,
        // VID[1] pin falling edge interrupt.
    GFVX_VID2PIN_FALLING_EDGE_INTR =           0x40004,
        // VID[2] pin falling edge interrupt.
```

GFVX_INTERRUPT_TYPE Enumeration Type (continued)

```
    GFVX_VID3PIN_FALLING_EDGE_INTR =           0x80008,
        // VID[3] pin falling edge interrupt.
    GFVX_VID4PIN_FALLING_EDGE_INTR =          0x100010,
        // VID[4] pin falling edge interrupt.
    GFVX_VID5PIN_FALLING_EDGE_INTR =          0x200020,
        // VID[5] pin falling edge interrupt.
    GFVX_VID6PIN_FALLING_EDGE_INTR =          0x400040,
        // VID[6] pin falling edge interrupt.
    GFVX_VID7PIN_FALLING_EDGE_INTR =          0x800080,
        // VID[7] pin falling edge interrupt.

    GFVX_VHSYNC_FALLING_EDGE_INTR =         0x2000200,
        // VHSYNC pin falling edge interrupt.
    GFVX_VVSYNC_FALLING_EDGE_INTR =           0x4000400
        // VVSYNC pin falling edge interrupt.
} GFVX_INTERRUPT_TYPE;
```

GFVXATTRIBUTES

This data type is used by "GFVxGetAttribute()" on page 231.

GFVXATTRIBUTES Enumeration Type

```
typedef enum _GFVXATTRIBUTES {
    GFVX_ATTR_VIDEO_BUF
} GFVXATTRIBUTES;
```

GFVxAPI Programming Sequence

The easiest way to learn the GFVxAPI is by referring to the demonstration application source code that come with the GFVxAPI package. A new application can be started by creating a project at the same directory level as the demo application source directory. This way the same directory tree structure can be maintained. The general sequence for programming the GFVxAPI is as follows:

1. Make sure that the build project or file has the right paths to the library and header files. Use relative directory addressing if possible. The **`GFSDK\Inc`** directory must be included.
2. Include `GFVx.h` in the source file. This header is all that is needed to access the exposed API functions.
3. Call **`GFRmOpen()`** before any other GFSDK functions (including the GFVxAPI).
4. Use the handle returned from the Resource Manager in the previous step to call **`GFRmComponentGet()`** with **`GF_VXAPI`** as the **`ComponentType`** in the **`GFRMCOMPONENT`** structure. The returned handle is the one to use throughout all GFVxAPI calls.
5. To better utilize GFVxAPI functions, it is a good idea to call **`GFVxGetProperty()`** to determine the version that you are using and the available video memory.
6. Make calls to the appropriate GFVxAPI functions. Always remember to pass the same **`GF_HANDLE`** returned from **`GFRmComponentGet()`**.
7. When all the GFVxAPI calls are completed, call **`GFRmComponentRelease()`**.
8. When it is time to exit the application, call **`RmClose()`** to release everything associated with the GFSDK.

CHAPTER

9

Multimedia Processor API (GFMmProcAPI)

Overview

This document describes the programming interface for the NVIDIA multimedia processor for handheld products. The interface provides a set of interfaces that can be used to allocate processor resources for use by applications. The interface is structured to consume resources only for the desired functionality.

The multimedia processor API comprises the following groups of functions:

Basic Audio

Basic Audio Overview

The initial function of the multimedia processor is to support basic audio functionality. It provides the ability to mix audio signals encoded from multiple sources to a single stream for the rendering device. It also provides an interface to record audio signals from captured from analog input sources. The main object of this layer is the manager. It the access point for obtaining system dependent resources such as players for multimedia processing. The Player controls the rendering of the data and may generate asynchronous events that may be monitored by the client by implementing the player listener interface. Player progress can be monitored and synchronized with other players via the time base interface. The final interface is the controllable interface. It can be used for obtaining one or more control objects.

Control

A control object is used to control some media processing functions. The set of operations are usually functionally related. Thus a control object provides a logical grouping of media processing functions. Controls are obtained from controllable. The player interface extends controllable. Therefore a player implementation can use the Control interface to extend its media processing functions. For example, a player can expose a volume control to allow the volume level to be set.

Multiple controls can be implemented by the same object. For example, an object can implement both volume control and tone control. In this case, the object can be used for controlling both the volume and tone generation.

For Basic Audio, no controls are defined.

Controllable

The controllable interface provides an abstract interface for obtaining the controls from an object like a Player. It provides methods to query all the supported controls and to obtain a particular control based on its class name.

- “GFMmProcControllableGetControl()” on page 253
- “GFMmProcControllableGetControls()” on page 253

Manager

Manager is the access point for obtaining system dependent resources such as players for multimedia processing. A player is an object used to control and render media that is specific to the content type of the data. Manager provides access to an implementation specific mechanism for constructing players. For convenience, Manager also provides a simplified method to generate simple tones.

Player

Player controls the rendering of time based media data. It provides the methods to manage the player's life cycle, controls the playback progress, obtains the presentation components, controls and provides the means to synchronize with other players.

- “GFMmProcPlayerSetTimeBase()” on page 268
- “GFMmProcPlayerStart()” on page 268
- “GFMmProcPlayerStop()” on page 269

Player Listener

Player Listener is the interface for receiving asynchronous events generated by players. Applications may implement this interface and register their implementations with the GFMmProcPlayerAddPlayerListener method in player.

A number of standard player events are defined here in this interface. Event types are defined as strings to support extensibility as different implementations may introduce proprietary events by adding new event types. To avoid name conflicts, proprietary events should be named with the "reverse domain name" convention. For example, a company named "mycompany" should name its proprietary event names with strings like "com.mycompany.myEvent" etc.

- “GFMmProcPlayerListenerPlayerUpdate()” on page 272

Time Base

A time base is a constantly ticking source of time. It measures the progress of time and provides the basic means for synchronizing media playback for Players. A time base measures time in microseconds in order to provide the necessary resolution for synchronization. It is acknowledged that some implementations may not be able to support time resolution in the microseconds range. For such implementations, the internal representation of time can be done within their limits. But the time reported via the API must be scaled to the microseconds range.

GFMmProcManagerGetSystemTimeBase provides the default time base used by the system.

- “GFMmProcTimeBaseGetTime()” on page 273

Basic Audio Interface Details

GFMmProcControllableGetControl()

Obtains the object that implements the specified control interface.

If the specified control interface is not supported then null is returned. If the controllable object supports multiple objects that implement the same specified control interface, only one of them will be returned. To obtain all the controls of that type, get the list of the controls and then check the list for the desired type.

Function Prototype

```
GFMMPROC_CONTROL GFMmProcControllableGetControl
(
    GFMMPROC_CONTROLLABLE handle,
    char*  controlType
)
```

Parameters

`handle`	The handle of the controllable object.
`controlType`	The class name of the control. The class name should be given either as the fully-qualified name of the class.

Returns: The object that implements the control, or null.

GFMmProcControllableGetControls()

Obtains the collection of controls from the object that implements this interface.

The list of control objects returned will not contain any duplicates. The list will not change over time. If no control is supported, a null is returned.

Function Prototype

```
GFMMPROC_CONTROL* GFMmProcControllableGetControls
(
    GFMMPROC_CONTROLLABLE handle,
    char* controlType
)
```

Parameters

handle	The handle of the controllable object.

Returns: The collection of control objects.

GFMMPROC_MANAGER Fields

GFMMPROC_MANAGER_MIDI_DEVICE_LOCATOR

The locator to create a MIDI player which gives access to the MIDI device by making MIDI control available. For example:

```
GFMMPROC_MANAGER m;
GFMMPROC_PLAYER p;
GFMMPROC_CONTROL c;
GFMMPROC_CONTOL_MIDI mc;

p = GFMmProcManagerCreatePlayer(m,
    GFMMPROC_MANAGER_MIDI_DEVICE_LOCATOR);
GFMmProcPlayerPrefetch(p); // opens the MIDI device
c = GFMmProcControllableGetControl(p,
    GFMMPROC_CONTROL_MIDI_STR);
mc = (GFMMPROC_CONTROL_MIDI)c;
```

The MIDI player returned does not carry any media data.
GFMmProcPlayerGetDuration returns 0 for this player. The content type of the player created from this locator is audio/midi.
A player for this locator may not be supported for all implementations.

GFMMPROC_MANAGER_TONE_DEVICE_LOCATOR

The locator to create a tone player to play back tone sequences. For example:

```
GFMMPROC_MANAGER m;
GFMMPROC_PLAYER p;
GFMMPROC_CONTROL c;
GFMMPROC_CONTOL_TONE tc;

p = GFMmProcManagerCreatePlayer(m,
    GFMMPROC_MANAGER_TONE_DEVICE_LOCATOR);
GFMmProcPlayerRealize(p);
c = GFMmProcControllableGetControl(p,
    GFMMPROC_CONTROL_TONE_STR);
tc = (GFMMPROC_CONTROL_TONE)c;
GFMmProcControlToneSetSequence(tc, mySequence);
GFMmProcPlayerStart(p);
```

If a tone sequence is not set on the tone player via its tone control, the player does not carry any sequence. GFMmProcPlayerGetDuration returns 0 for this player. The content type of the player created from this locator is audio/x-tone-seq.

A player for this locator may not be supported for all implementations.

GFMmProcManagerCreatePlayer()

This function creates a player from an input locator.

Function Prototype

```
GFMMPROC_PLAYER GFMmProcManagerCreatePlayer
(
    GFMMPROC_MANAGER handle,
    char* locator
)
```

Parameters

handle	The handle of the controllable object.
locator	A locator string in URI syntax that describes the media content.

Returns: A new Player.

GFMmProcManagerCreatePlayerForDataSource()

This function creates a player for a data source.

Function Prototype

```
GFMMPROC_PLAYER GFMmProcManagerCreatePlayerForDataSource
(
    GFMMPROC_MANAGER handle,
    DataSource source
)
```

Parameters

handle	The handle of the manager object.
source	The data source that provides the media content.

Returns: A new player.

GFMmProcManagerCreatePlayerForStream()

Create a player to play back media from an input stream.

The type argument specifies the content-type of the input media. If null is given, manager will attempt to determine the type. However, since determining the media type is non-trivial for some media types, it may not be feasible in some cases.

Function Prototype

```
GFMMPROC_PLAYER GFMmProcManagerCreatePlayerForStream
(
    GFMMPROC_MANAGER handle,
    GFMMPROC_STREAM stream,
    char* type
)
```

Parameters

handle	The handle of the manager object.
stream	The stream that delivers the input media.
type	The content type of the media.

Returns: A new player.

GFMmProcManagerGetSupportedContentTypes()

Returns the list of supported content types for the given protocol.

See content types for the syntax of the content types returned. See protocol name for the syntax of the protocol used. For example, if the given protocol is "http", then the supported content types that can be played back with the http protocol will be returned. If null is passed in as the protocol, all the supported content types for this implementation will be returned. The returned array must be non-empty. If the given protocol is an invalid or unsupported protocol, then an empty array will be returned.

Function Prototype

```
char* GFMmProcManagerGetSupportedContentTypes
(
    GFMMPROC_MANAGER handle,
    char* protocol
)
```

Parameters

handle	The handle of the manager object.
protocol	The input protocol for the supported content types.

Returns: The list of supported content types for the given protocol.

GFMmProcManagerGetSupportedProtocols()

Returns the list of supported protocols given the content type.

The protocols are returned as strings which identify what locators can be used for creating players. See protocol name for the syntax of the protocols returned. See content types for the syntax of the content type used. For example, if the given content type is "audio/x-wav", then the supported protocols that can be used to play back audio/x-wav will be returned. If null is passed in as the content type, all the supported protocols for this implementation will be returned. The returned array must be non-empty. If

the given content type is an invalid or unsupported content type, then an empty array will be returned.

Function Prototype

```
char* GFMmProcManagerGetSupportedProtocols
(
    GFMMPROC_MANAGER handle,
     char* contentType
)
```

Parameters

handle	The handle of the manager object.
contentType	The content type for the supported protocols.

Returns: The list of supported protocols for the given content type.

GFMmProcManagerGetSystemTimeBase()

Get the time-base object for the system.

Function Prototype

```
GFMMPROC_TIME_BASE GFMmProcManagerGetSystemTimeBase
(
    GFMMPROC_MANAGER handle
)
```

Parameters

handle	The handle of the manager object.

Returns: The system time base.

GFMmProcManagerPlayTone()

Play back a tone as specified by a note and its duration. A note is given in the range of 0 to 127 inclusive. The frequency of the note can be calculated from the following formula:

```
SEMITONE_CONST = 17.31234049066755 = 1/(ln(2^(1/12)))
note = ln(freq/8.176)*SEMITONE_CONST
(The musical note A = MIDI note 69 (0x45) = 440 Hz.)
```

This call is a non-blocking call. Notice that this method may utilize CPU resources significantly on devices that don't have hardware support for tone generation.

Function Prototype

```
void GFMmProcManagerPlayTone
(
    GFMMPROC_MANAGER handle,
    NvU32 note,
    NvU32 duration,
    NvU32 volume
)
```

Parameters

handle	The handle of the manager object.
note	Defines the tone of the note as specified by the above formula.
duration	The duration of the tone in milli-seconds. Duration must be positive.
volume	Audio volume range from 0 to 100. 100 represents the maximum volume at the current hardware level. Setting the volume to a value less than 0 will set the volume to 0. Setting the volume to greater than 100 will set the volume to 100.

GFMmProcManagerProcess()

This function signals the audio engine to check if there are any status updates from the hardware or flush any cached updates to hardware. Normal operation does not require this level of interaction as the audio engine will periodically do updates internally. However, this function may be called to force the engine to make them.

Function Prototype

```
void GFMmProcManagerProcess
(
    GFMMPROC_MANAGER handle
)
```

Parameters

handle	The handle of the manager object.

GFMMPROC_PLAYER Fields

GFMMPROC_PLAYER_CLOSED
The state of the player indicating that the player is closed.

GFMMPROC_PLAYER_PREFETCHED
The state of the player indicating that it has acquired all the resources to begin playing.

GFMMPROC_PLAYER_REALIZED
The state of the player indicating that it has acquired the required information but not the resources to function.

GFMMPROC_PLAYER_STARTED
The state of the player indicating that the player has already started.

GFMMPROC_PLAYER_TIME_UNKNOWN
The returned value indicating that the requested time is unknown.

GFMMPROC_PLAYER_UNREALIZED
The state of the player indicating that it has not acquired the required information and resources to function.

GFMmProcPlayerAddPlayerListener()

Add a player listener for this player.

Function Prototype

```
void GFMmProcPlayerAddPlayerListener
(
    GFMMPROC_PLAYER handle
```

Function Prototype

```
    GFMMPROC_PLAYER_LISTENER playerListener
)
```

Parameters

handle	The handle of the player object.
playerListener	The listener to add. If null is used, the request will be ignored.

GFMmProcPlayerClose()

Close the player and release its resources.

When the method returns, the player is in the CLOSED state and can no longer be used. A CLOSED event will be delivered to the registered player listeners. If close is called on a closed player the request is ignored.

Function Prototype

```
void GFMmProcPlayerClose
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

GFMmProcPlayerDeallocate()

Release the scarce or exclusive resources like the audio device acquired by the Player.

When GFMmProcPlayerDeallocate returns, the player is in the UNREALIZED or REALIZED state. If the player is blocked at the realize call while realizing, calling GFMmProcPlayerDeallocate unblocks the realize call and returns the player to the UNREALIZED state. Otherwise, calling GFMmProcPlayerDeallocate returns the player to the REALIZED state. If GFMmProcPlayerDeallocate is called when the player is in the UNREALIZED or REALIZED state, the request is ignored. If the player is

STARTED when GFMmProcPlayerDeallocate is called, GFMmProcPlayerDeallocate will implicitly call stop on the player.

Function Prototype

```
void GFMmProcPlayerDeallocate
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

GFMmProcPlayerGetContentType()

Get the content type of the media that's being played back by this player.

See content type for the syntax of the content type returned.

Function Prototype

```
char* GFMmProcPlayerGetContentType
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

Returns: The content type being played back by this player.

GFMmProcPlayerGetDuration()

Get the duration of the media. The value returned is the media's duration when played at the default rate. If the duration cannot be determined (for example, the player is presenting live media) GFMmProcPlayerGetDuration returns GFMMPROC_PLAYER_TIME_UN-KNOWN.

Function Prototype

```
NvS64 GFMmProcPlayerGetDuration
(
```

Function Prototype

```
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player objec.

Returns: The duration in microseconds or GFMMPROC_PLAYER_TIME_UNKNOWN.

GFMmProcPlayerGetMediaTime()

Gets this player's current media time.

GFMmProcPlayerGetMediaTime may return GFMMPROC_PLAYER_TIME_UNKNOWN to indicate that the media time cannot be determined. However, once GFMmProcPlayerGetMediaTime returns a known time (time not equals to GFMMPROC_PLAY-ER_TIME_UNKNOWN), subsequent calls to GFMmProcPlayerGetMediaTime must not return GFMMPROC_PLAYER_TIME_UNKNOWN.

Function Prototype

```
NvS64 GFMmProcPlayerGetMediaTime
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

Returns: The current media time in microseconds or GFMMPROC_PLAYER_TIME_UN-KNOWN.

GFMmProcPlayerGetState()

Gets the current state of this player. The possible states are: UNREALIZED, REALIZED, PREFETCHED, STARTED, CLOSED.

Function Prototype

```
NvS32 GFMmProcPlayerGetState
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

Returns: The player's current state.

GFMmProcPlayerGetTimeBase()

Gets the time base that this player is using.

Function Prototype

```
GFMMPROC_TIME_BASE GFMmProcPlayerGetTimeBase
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

Returns: The time base that this player is using.

GFMmProcPlayerPrefetch()

Acquires the scarce and exclusive resources and processes as much data as necessary to reduce the start latency.

When GFMmProcPlayerPrefetch completes successfully, the player is in the PREFETCHED state. If GFMmProcPlayerPrefetch is called when the player is in the UNREALIZED state, it will implicitly call realize. If GFMmProcPlayerPrefetch is called when the player is already in the PREFETCHED state, the player may still process data necessary to reduce the

start latency. This is to guarantee that start latency can be maintained at a minimum. GFMmProcPlayerPrefetch may be called when the player is in the STARTED state to signal that more data is available.

If the player cannot obtain all of the resources it needs, the player will not be able to start. However, GFMmProcPlayerPrefetch may be called again when the needed resource is later released perhaps by another Player or application.

Function Prototype

```
void GFMmProcPlayerPrefetch
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

GFMmProcPlayerPull()

Prevents the player from periodically pulling data. This is used when the client wants explicit control over when the data can be mapped by the hardware. The default state of the player is true.

Function Prototype

```
void GFMmProcPlayerPull
(
    GFMMPROC_PLAYER handle,
    NvS32 allow
)
```

Parameters

handle	The handle of the player object.
allow	When set to true, data will be periodically pulled into the player.

GFMmProcPlayerRealize()

Constructs portions of the player without acquiring the scarce and exclusive resources. This may include examining media data and may take some time to complete. When realize completes successfully, the player is in the

REALIZED state. If realize is called when the player is in the REALIZED, PREFETCHTED or STARTED state, the request will be ignored.

Function Prototype

```
void GFMmProcPlayerRealize
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

GFMmProcPlayerRemovePlayerListener()

Remove a player listener for this player.

Function Prototype

```
void GFMmProcPlayerRemovePlayerListener
(
    GFMMPROC_PLAYER handle
    GFMMPROC_PLAYER_LISTENER playerListener
)
```

Parameters

handle	The handle of the player object.
playerListener	The listener to remove. If null is used or the given player listener is not a listener for this player, the request will be ignored.

GFMmProcPlayerSetLoopCount()

Set the number of times the player will loop and play the content.

By default, the loop count is one. That is, once started, the player will start playing from the current media time to the end of media once. If the loop count is set to N where N is bigger than one, starting the player will start playing the content from the current media time to the end of media. It will then loop back to the beginning of the content (media time zero) and play till the end of the media. The number of times it will loop to the beginning and play to the end of media will be N-1.

- Setting the loop count to 0 is invalid.
- Setting the loop count to -1 will loop and play the content indefinitely.

If the player is stopped before the preset loop count is reached either because stop is called or a preset stop time (set with the stop time control) is reached, calling start again will resume the looping playback from where it was stopped until it fully reaches the preset loop count.

An END_OF_MEDIA event will be posted every time the player reaches the end of media. If the player loops back to the beginning and starts playing again because it has not completed the loop count, a STARTED event will be posted.

Function Prototype

```
void GFMmProcPlayerSetLoopCount
(
    GFMMPROC_PLAYER handle
    NvS32 count
)
```

Parameters

handle	The handle of the player object.
count	Indicates the number of times the content will be played. 1 is the default. 0 is invalid. -1 indicates looping indefinitely.

GFMmProcPlayerSetMediaTime()

Sets the player's media time.

For some media types, setting the media time may not be very accurate. The returned value will indicate the actual media time set. Setting the media time to negative values will effectively set the media time to zero. Setting the media time to beyond the duration of the media will set the time to the end of media. There are some media types that cannot support the setting of media time.

Function Prototype

```
NvS64 GFMmProcPlayerSetMediaTime
(
```

Function Prototype

```
    GFMMPROC_PLAYER handle
    NvS64 now
)
```

Parameters

handle	The handle of the player object.
now	The new media time in microseconds.

Returns: The actual media time set in microseconds.

GFMmProcPlayerSetTimeBase()

Sets the time base for this player.

A player has a default time base that is determined by the implementation. To reset a player to its default time base, call GFMmProcPlayerSetTimeBase(null).

Function Prototype

```
void GFMmProcPlayerSetTimeBase
(
    GFMMPROC_PLAYER handle,
    GFMMPROC_TIME_BASE master
)
```

Parameters

handle	The handle of the player object.
master	The new time base or null to reset the player to its default time base.

GFMmProcPlayerStart()

Starts the player as soon as possible. If the player was previously stopped by calling stop or reaching a preset stop time, it will resume playback from where it was previously stopped. If the player has reached the end of media, calling start will automatically start the playback from the start of the media.

When start returns successfully, the player must have been started and a STARTED event will be delivered to the registered player's listener. However, the Player is not guaranteed to be in the STARTED state. The

Player may have already stopped (in the PREFETCHED state) because the media has 0 or a very short duration.

If start is called when the Player is in the UNREALIZED or REALIZED state, it will implicitly call prefetch. If start is called when the Player is in the STARTED state, the request will be ignored.

Function Prototype

```
void GFMmProcPlayerStart
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

GFMmProcPlayerStop()

Stops the player. It will pause the playback at the current media time.

When stop returns, the player is in the PREFETCHED state. A STOPPED event will be delivered to the registered player's listeners. If stop is called on a stopped player, the request is ignored.

Function Prototype

```
void GFMmProcPlayerStop
(
    GFMMPROC_PLAYER handle
)
```

Parameters

handle	The handle of the player object.

GFMMPROC_PLAYER_LISTENER Fields

GFMMPROC_PLAYER_LISTENER_BUFFERING_STARTED

Posted when the player enters into a buffering mode. Applications may require this event to handle other tasks. When this event is received, the event data parameter will be an NvS32 object designating the media time when the buffering is started.

GFMMPROC_PLAYER_LISTENER_BUFFERING_STOPPED

Posted when the player leaves the buffering mode. Applications may require this event to handle other tasks. When this event is received, the event data parameter will be an NvS32 object designating the media time when the buffering stopped.

GFMMPROC_PLAYER_LISTENER_CLOSED

Posted when a player is closed. When this event is received, the event data parameter is null.

GFMMPROC_PLAYER_LISTENER_END_OF_MEDIA

Posted when a player has reached the end of the media. When this event is received, the event data parameter will be an NvS32 object designating the media time when the player reached end of media and stopped.

GFMMPROC_PLAYER_LISTENER_DEVICE_AVAILABLE

Posted when the system or another higher priority application has released an exclusive device which is now available to the player. The player will be in the REALIZED state when this event is received. The application may acquire the device with the prefetch or start method. A DEVICE_UNAVAILABLE event must precede this event.

The event data parameter is a string specifying the name of the device.

GFMMPROC_PLAYER_LISTENER_DEVICE_UNAVAILABLE

Posted when the system or another higher priority application has temporarily taken control of an exclusive device which was previously available to the Player. The Player will be in the REALIZED state when this event is received. This event must be followed by either a DEVICE_AVAILABLE event when the device becomes available again or an ERROR event if the device becomes permanently unavailable.

The event data parameter is a String specifying the name of the device.

GFMMPROC_PLAYER_LISTENER_DURATION_UPDATED
Posted when the duration of a Player is updated. This happens for some media types where the duration cannot be derived ahead of time. It can only be derived after the media is played for a period of time – for example, when it reaches a key frame with duration info; or when it reaches the end of media. When this event is received, the event data parameter will be an NvS32 object designating the duration of the media.

GFMMPROC_PLAYER_LISTENER_ERROR
Posted when an error had occurred. When this event is received, the event data parameter will be a string object specifying the error message.

GFMMPROC_PLAYER_LISTENER_RECORD_ERROR
Posted when an error occurs during the recording. The current recording will be discarded. The application may set a new record location or stream to start recording again. When this event is received, the event data parameter will be a string object specifying the error message.

GFMMPROC_PLAYER_LISTENER_RECORD_STARTED
Posted when recording is started. When this event is received, the event data parameter will be an NvS32 object designating the media time when the recording is started.

GFMMPROC_PLAYER_LISTENER_RECORD_STOPPED

Posted when recording is stopped. When this event is received, the event data parameter will be an NvS32 object designating the media time when the recording stopped.

GFMMPROC_PLAYER_LISTENER_STARTED
Posted when a player is started. When this event is received, the event data parameter will be an NvS32 object designating the media time when the player is started.

GFMMPROC_PLAYER_LISTENER_STOPPED
Posted when a player stops in response to the stop method call. When this event is received, the event data parameter will be an NvS32 object designating the media time when the player stopped.

GFMMPROC_PLAYER_LISTENER_STOPPED_AT_TIME
Posted when a player is stopped as responding to the GFMmProcSetStopTime call using the GFMmProcControlStopTime. When this event is received, the event data parameter will be an NvS32 object designating the media time when the player is stopped.

GFMMPROC_PLAYER_LISTENER_VOLUME_CHANGED
Posted when the volume of an audio device is changed. When this event is received, the event data parameter will be a volume control object. The new volume can be queried from the volume control.

GFMmProcPlayerListenerPlayerUpdate()

This method is called to deliver an event to a registered listener when a player event is observed.

Function Prototype

```
void GFMmProcPlayerListenerPlayerUpdate
(
    GFMMPROC_PLAYER player,
    char* event,
```

Function Prototype

```
    void* eventData
)
```

Parameters

handle	The handle of the player listener object.
player	The player which generated the event.
event	The event generated as defined by the enumerated types.
eventData	The associated event data.

GFMmProcTimeBaseGetTime()

Get the current time of this time base. The values returned must be non-negative and non-decreasing over time.

Function Prototype

```
NvS32 GFMmProcTimeBaseGetTime
(
    GFMMPROC_TIME_BASE handle
)
```

Parameters

handle	The handle of the time base object.

Returns: The current time base time in microseconds.

Basic Audio Controls

The next layer of audio functionality supported by the NVIDIA multimedia processor adds a basic set of control-type objects. The types of controls supported by this layer have interfaces for MIDI, pitch, rate, record, customizing the stop of a player, tempo for MIDI sequences, and volume. The MIDI control interface provides access to MIDI rendering and transmitting devices. The pitch control interface raises or lowers the playback pitch of audio without changing the playback speed. The rate control interface controls the playback rate of a player without affecting the pitch of the content. The record control interface controls the recording of media from a player to an output stream and terminates the transform line. The stop time control interface allows the player to be stopped at a specified time. The tempo control interface is derived from the rate control interface to modify the tempo of a MIDI file in the same manner as a tempo event in a MIDI sequence. This effect is temporary if the MIDI sequence contains tempo events. The overall playback rate of a MIDI sequence is controlled via the GFMmProcControlRateSetRate method of the base class. The tone control interface enables the playback of a user-defined monotonic tone sequence. The volume control interface allows the manipulation of the audio volume of a player.

MIDI Control

MIDI Control provides access to MIDI rendering and transmitting devices.

Pitch Control

Pitch control raises or lowers the playback pitch of audio without changing the playback speed. Pitch control can be implemented in players for MIDI media or sampled audio. It is not possible to set audible output to an absolute pitch value. This control raises or lowers pitch relative to the original.

- "GFMmProcControlPitchGetMaxPitch()" on page 284
- "GFMmProcControlPitchGetMinPitch()" on page 285
- "GFMmProcControlPitchGetPitch()" on page 285
- "GFMmProcControlPitchSetPitch()" on page 285

Rate Control

Rate control controls the playback rate of a Player. The rate defines the relationship between the Player's media time and its time base. Rates are specified in "milli- percentage". For example, a rate of 200'000 indicates that media time will pass twice as fast as the time base time once the Player starts. Similarly, a negative rate indicates that the Player runs in the opposite direction of its time base, i.e. playing in reverse.

All Players must support the default rate 100'000. Players that support only the default rate must not implement this interface. Players that support other rates besides 100'000, should implement this interface and specify the appropriate minimum and maximum playback rates.

For audio, specific implementations may change the playback pitch when changing the playback rate. This may be viewed as an undesirable side-effect. See pitch control for changing pitch without changing playback rate.

- "GFMmProcControlRateGetMaxRate()" on page 286
- "GFMmProcControlGetMinRate()" on page 286
- "GFMmProcControlGetRate()" on page 287
- "GFMmProcControlSetRate()" on page 287

Record Control

Record control controls the recording of media from a player. Record control records what's currently being played by the player.

- "GFMmProcControlRecordCommit()" on page 288

Stop Time Control

Stop time control allows one to specify a preset stop time for a player.

Tempo Control

Tempo control controls the tempo, in musical terms, of a song. Tempo control is typically implemented in players for MIDI media, i.e. playback of a Standard MIDI File (SMF). Tempo control is basic functionality for a MIDI playback application. This is in contrast to MIDI control, which targets advanced applications. Moreover, tempo control needs a sequence - e.g. a MIDI file - to operate. MIDI control does not require a sequence.

Tone Control

Tone control is the interface to enable playback of a user-defined monotonic tone sequence. A tone sequence is specified as a list of tone-duration pairs and user-defined sequence blocks. The list is packaged as an array of bytes. The GFMmProcControlToneSetSequence method is used to input the sequence to the tone control.

Volume Control

Volume control is an interface for manipulating the audio volume of a player.

Basic Audio Controls Interface Details

GFMMPROC_CONTROL Fields

GFMMPROC_CONTROL_MIDI_CONTROL_CHANGE
Command value for Control Change message (0xB0, or 176).

GFMMPROC_CONTROL_MIDI_NOTE_ON
Command value for Note On message (0x90, or 144).

GFMmProcControlMidiGetBankList()

Returns list of installed banks. If the custom parameter is true, a list of custom banks is returned. Otherwise, a list of all banks (custom and internal) is returned. As there is no MIDI equivalent to this method, this method is optional, indicated by GFMmProcControlMidiIsBankQuerySupported. If it returns false, this function is not supported.

Function Prototype

```
int* GFMmProcControlMidiGetBankList
(
    GFMMPROC_CONTROL_MIDI handle,
    NvS32 custom
)
```

Parameters

handle	The handle of the MIDI control object.
custom	If set to true, returns list of custom banks.

GFMmProcControlMidiGetChannelVolume()

Get the volume for the given channel. The return value is independent of the master volume, which is set and retrieved with volume control. As there is no MIDI equivalent to this method, the implementation may not always know the current volume for a given channel. In this case the return value is -1.

Function Prototype

```
int* GFMmProcControlMidiGetChannelVolume
(
    GFMMPROC_CONTROL_MIDI handle,
    NvS32 channel
)
```

Parameters

handle	The handle of the MIDI control object.
channel	The range is 0..15.

Returns: The channel volume of 0..127, or -1 if not known.

GFMmProcControlMidiGetKeyName()

Given bank, program and key, get name of key. This method applies to key-mapped banks (i.e. percussive banks or effect banks) only. A return value of null means that the specified key is not mapped to a sound. For melodic banks, where each key (=note) produces the same sound at different pitch, this method always returns null. For space-saving reasons, an implementation may return an empty string instead of the key name. To find out which keys in a specific program are mapped to a sound, iterate through all keys (0-127) and compare the return value of GFMmProcControlMidiGetKeyName to non-null.

As there is no MIDI equivalent to this method, this method is optional, indicated by GFMmProcControlMidiIsBankQuerySupported. If it returns false, this function is not supported and throws an exception.

Function Prototype

```
int* GFMmProcControlMidiGetKeyName
(
    GFMMPROC_CONTROL_MIDI handle,
```

Function Prototype

```
    NvS32 bank,
    NvS32 program,
    NvS32 key
)
```

Parameters

`handle`	The handle of the MIDI control object.
`bank`	The range is 0..16383.
`program`	The range is 0..127.
`key`	The range is 0..127.

Returns: The name of the specified key, empty string, or null if the key is not mapped to a sound.

GFMmProcControlMidiGetProgram()

Returns program assigned to channel. It represents the current state of the channel. During playback of a MIDI file, the program may change due to program change events in the MIDI file.

To set a program for a channel, use GFMmProcControlMidiSetProgram(int, int, int).

The returned array is represented by an array {bank,program}.

If the device has not been initialized with a MIDI file, or the MIDI file does not contain a program change for this channel, an implementation specific default value is returned.

As there is no MIDI equivalent to this method, this method is optional, indicated by GFMmProcControlMidiIsBankQuerySupported. If it returns false, this function is not supported.

Function Prototype

```
int* GFMmProcControlMidiGetProgram
(
    GFMMPROC_CONTROL_MIDI handle,
    NvS32 channel
)
```

Parameters

handle	The handle of the MIDI control object.
channel	The range is 0..15.

GFMmProcControlMidiGetProgramName()

Given bank and program, get name of program. For space-saving reasons, an implementation may return an empty string.

As there is no MIDI equivalent to this method, this method is optional, indicated by GFMmProcControlMidiIsBankQuerySupported. If it returns false, this function is not supported.

Function Prototype

```
int* GFMmProcControlMidiGetProgramName
(
    GFMMPROC_CONTROL_MIDI handle,
    NvS32 bank,
    NvS32 program
)
```

Parameters

handle	The handle of the MIDI control object.
bank	The range is 0..16383.
program	The range is 0..127.

GFMmProcControlMidiIsBankQuerySupported()

Returns whether banks of the synthesizer can be queried. If this functions returns non-zero, then the following methods can be used to query banks:

- GFMmProcControlMidiGetBankList
- GFMmProcControlMidiGetKeyName
- GFMmProcControlMidiGetProgram
- GFMmProcControlMidiGetProgramList
- GFMmProcControlMidiGetProgramName

A program refers to a single instrument. This is also known as a patch.

Function Prototype

```
NvS32 GFMmProcControlMidiIsBankQuerySupported
(
    GFMMPROC_CONTROL_MIDI handle,
)
```

Parameters

handle	The handle of the MIDI control object.

Returns: Non-zero if the querying of banks is supported.

GFMmProcControlMidiLongMidiEvent()

Sends a long MIDI event to the device, typically a system exclusive message. This method passes the data directly to the receiving device.

The data array's contents are not checked for validity.

It is possible to send short events or even a series of short events with this method.

Function Prototype

```
int* GFMmProcControlMidiLongMidiEvent
(
    GFMMPROC_CONTROL_MIDI handle,
    char* data,
    NvS32 offset,
    NvS32 length
)
```

Parameters

handle	The handle of the MIDI control object.
data	The array of the bytes to send.
offset	The start offset in data array.
length	The number of bytes to be sent.

Returns: The number of bytes actually sent to the device or -1 if an error occurred.

GFMmProcControlMidiSetChannelVolume()

Set volume for the given channel. To mute, set to 0. This sets the current volume for the channel and may be overwritten during playback by events in a MIDI sequence.

It is a high level convenience function.

The channel volume is independent of the master volume, which is accessed with volume control. Setting the channel volume does not modify the value of the master volume - and vice versa: changing the value of master volume does not change any channel's volume value. The synthesizer mixes the output of up to 16 channels, each channel with its own channel volume. The master volume then controls the volume of the mix. Consequently, the effective output volume of a channel is the product of master volume and channel volume.

Setting the channel volume does not generate a GFMMPROC_PLAYER_LISTENER_VOL-UME_CHANGED event.

Function Prototype

```
int* GFMmProcControlMidiSetChannelVolume
(
    GFMMPROC_CONTROL_MIDI handle,
    NvS32 channel,
    NvS32 volume
)
```

Parameters

handle	The handle of the MIDI control object.
channel	The range is 0..15.
volume	The range is 0..127.

GFMmProcControlMidiSetProgram()

Set program of a channel. This sets the current program for the channel and may be overwritten during playback by events in a MIDI sequence.

It is a high level convenience function.

In order to use the default bank (the initial bank), set the bank parameter to -1. In order to set a program without explicitly setting the bank, use the following call:

```
NvS32 PROGRAM_CHANGE = 0xC0;
GFMmProceControlMidiShortMidiEvent
    (PROGRAM_CHANGE | channel, program, 0);
```

Function Prototype

```
int* GFMmProcControlMidiSetProgram
(
    GFMMPROC_CONTROL_MIDI handle,
    NvS32 channel,
    NvS32 bank,
    NvS32 program
)
```

Parameters

handle	The handle of the MIDI control object.
channel	The range is 0..15.
bank	The range is 0..16383, or -1 for default bank.
program	The range is 0..127

GFMmProcControlMidiShortMidiEvent()

Sends a short MIDI event to the device. Short MIDI events consist of 1, 2, or 3 unsigned bytes. For non-real time events, the first byte is split up into status (upper nibble, 0x80-0xF0) and channel (0x00-0x0F). For example, to send a Note On event on a given channel, use this line:

```
GFMmProceControlMidiShortMidiEvent
    (GFMMPROC_CONTROL_MIDI_NOTE_ON | channel, note, velocity);
```

For events with less than 3 bytes, set the remaining data bytes to 0.

There is no guarantee that a specific implementation of a MIDI device supports all event types. Also, the MIDI protocol does not implement flow control and it is not guaranteed that an event reaches the destination. In both these cases, this method fails silently.

Static error checking is performed on the passed parameters. They have to specify a valid, complete MIDI event. Events with type < 0x80 are not valid MIDI events (-> running status).

Function Prototype

```
int* GFMmProcControlMidiShortMidiEvent
(
    GFMMPROC_CONTROL_MIDI handle,
    NvS32 type,
    NvS32 data1,
    NvS32 data2
)
```

Parameters

handle	The handle of the MIDI control object.
type	The range is 0x80..0xFF, excluding 0xF0 and 0xF7, which are reserved for system exclusive
data1	For 2 and 3-byte events: first data byte is 0..127
data2	For 3-byte events: second data byte is 0..127

GFMmProcControlPitchGetMaxPitch()

Gets the maximum playback pitch raise supported by the player.

Function Prototype

```
NvS32 GFMmProcControlPitchGetMaxPitch
(
    GFMMPROC_CONTROL_PITCH handle
)
```

Parameters

handle	The handle of the pitch control object.

Returns: The maximum pitch raise in "milli-semitones".

GFMmProcControlPitchGetMinPitch()

Gets the minimum playback pitch raise supported by the player.

Function Prototype

```
NvS32 GFMmProcControlPitchGetMinPitch
(
    GFMMPROC_CONTROL_PITCH handle
)
```

Parameters

handle	The handle of the pitch control object.

Returns: The minimum pitch raise in "milli-semitones".

GFMmProcControlPitchGetPitch()

Gets the current playback pitch raise.

Function Prototype

```
NvS32 GFMmProcControlPitchGetPitch
(
    GFMMPROC_CONTROL_PITCH handle
)
```

Parameters

handle	The handle of the pitch control object.

Returns: The current playback pitch raise in "milli-semitones".

GFMmProcControlPitchSetPitch()

Sets the relative pitch raise. The pitch change is specified in "milli-semitones", i.e. 1000 times the number of semitones to raise the pitch. Negative values lower the pitch by the number of milli-semitones.

The GFMmProcControlPtchSetPitch method returns the actual pitch change set by the Player. Players should set their pitch raise as close to the requested value as possible, but are not required to set it to the exact value of any argument other than 0. A Player is only guaranteed to set its pitch change exactly to 0. If the given pitch raise is less than the value returned by

GFMmProcControlPitchGetMinPitch or greater than the value returned by GFMmProcControlPitchGetMaxPitch, it will be adjusted to the minimum or maximum supported pitch raise respectively.

Function Prototype

```
NvS32 GFMmProcControlPitchSetPitch
(
    GFMMPROC_CONTROL_PITCH handle,
    NvS32 millisemitones
)
```

Parameters

handle	The handle of the pitch control object.
millisemitones	The number of semi tones to raise the playback pitch. It is specified in "milli-semitones".

Returns: The actual pitch raise set in "milli-semitones".

GFMmProcControlRateGetMaxRate()

Gets the maximum playback rate supported by the Player.

Function Prototype

```
NvS32 GFMmProcControlRateGetMaxRate
(
    GFMMPROC_CONTROL_RATE handle
)
```

Parameters

handle	The handle of the rate control object.

Returns: The maximum rate in "milli-percentage".

GFMmProcControlGetMinRate()

Gets the minimum playback rate supported by the Player.

Function Prototype

```
NvS32 GFMmProcControlGetMinRate
(
```

Function Prototype

```
    GFMMPROC_CONTROL_RATE handle
)
```

Parameters

handle	The handle of the rate control object.

Returns: The minimum rate in "milli-percentage".

GFMmProcControlGetRate()

Gets the current playback rate.

Function Prototype

```
NvS32 GFMmProcControlGetRate
(
    GFMMPROC_CONTROL_RATE handle
)
```

Parameters

handle	The handle of the rate control object.

Returns: The current playback rate in "milli-percentage".

GFMmProcControlSetRate()

Sets the playback rate. The specified rate is 1000 times the percentage of the actual rate. For example, to play back at twice the speed, specify a rate of 200'000.

The setRate method returns the actual rate set by the Player. Player should set their rate as close to the requested value as possible, but are not required to set the rate to the exact value of any argument other than 100'000. A Player is only guaranteed to set its rate exactly to 100'000. If the given rate is less than GFMmProcControlGetMinRate or greater than GFMmProcControlGetMaxRate, the rate will be adjusted to the minimum or maximum supported rate respectively.

If the player is already started, GFMmProcControlSetRate will immediately take effect.

Function Prototype

```
NvS32 GFMmProcControlSetRate
(
    GFMMPROC_CONTROL_RATE handle,
    NvS32 millirate
)
```

Parameters

handle	The handle of the rate control object.
millirate	The playback rate to set. The rate is given in a "milli-percentage" value.

Returns: The actual rate set in "milli-percentage".

GFMmProcControlRecordCommit()

Complete the current recording. If the recording is in progress, commit will implicitly call GFMmProcControlRecordStopRecord. To record again after commit has been called, GFMmProcControlRecordSetRecordLocation or GFMmProcControlRecordSetRecordStream must be called.

Function Prototype

```
void GFMmProcControlRecordCommit
(
    GFMMPROC_CONTROL_RATE handle
)
```

Parameters

handle	The handle of the record control object.

GFMmProcControlRecordGetContentType()

Return the content type of the recorded media. The content type is given in the content type syntax.

Function Prototype

```
char* GFMmProcControlRecordGetContentType
(
    GFMMPROC_CONTROL_RATE handle
)
```

Parameters

handle	The handle of the record control object.

Returns: The content type of the media.

GFMmProcControlRecordSetRecordLocation()

Set the output location where the data will be recorded.

Whenever possible, the recording format is the same as the format of the input media. In some cases, the recording format may be different from the input format if the input format is not a recordable format, e.g. streaming media data. An application can query the recorded format by calling the GFMmProcPlayerGetContentType method.

Function Prototype

```
void GFMmProcControlRecordSetRecordLocation
(
    GFMMPROC_CONTROL_RATE handle,
    char* locator
)
```

Parameters

handle	The handle of the record control object.
locator	The locator specifying where the recorded media will be saved.

GFMmProcControlRecordSetRecordStream()

Set the output stream where the data will be recorded. Whenever possible, the recording format is the same as the format of the input media. In some

cases, the recording format may be different from the input format if the input format is not a recordable format, e.g. streaming media data. An application can query the recorded format by calling the GFMmProcPlayerGetContentType method.

Function Prototype

```
void GFMmProcControlRecordSetRecordStream
(
    GFMMPROC_CONTROL_RATE handle,
    GFMMPROC_STREAM stream.
    Char* type
)
```

Parameters

handle	The handle of the record control object.
stream	The output stream where the data will be recorded.
type	The content type of the media.

GFMmProcControlRecordSetRecordSizeLimit()

Set the record size limit. This limits the size of the recorded media to the number of bytes specified.

When recording is in progress, commit will be called implicitly in the following cases:

- Record size limit is reached.
- If the requested size is less than the already recorded size.
- No more space is available.

Once a record size limit has been set, it will remain so for future recordings until it is changed by another GFMmProcControlRecordSetRecordSizeLimit call.

To remove the record size limit, set it to MAX_VALUE. By default, the record size limit is not set. Only positive values can be set. Zero or negative values are invalid.

Function Prototype

```
NvS32 GFMmProcControlRecordSetRecordSizeLimit
(
```

Function Prototype

```
    GFMMPROC_CONTROL_RATE handle,
    NvS32 size
)
```

Parameters

handle	The handle of the record control object.
size	The record size limit in number of bytes.

Returns: The actual size limit set.

GFMmProcControlRecordStartRecord()

Start recording the media. If the Player is already started, GFMmProcControlRecordStartRecord will immediately start the recording. If the Player is not already started, GFMmProcControlRecordStartRecord will not record any media. It will put the recording in a "standby" mode. As soon as the Player is started, the recording will start right away.

If GFMmProcControlRecordStartRecord is called when the recording has already started, it will be ignored.

When GFMmProcControlRecordStartRecord returns, the recording has started and a GFMMPROC_PLAYER_LISTENER_RECORD_STARTED event will be delivered through the player listener. If an error occurs while recording is in progress, GFMMPROC_PLAYER_LISTENER_RECORD_ERROR event will be delivered via the player listener.

Function Prototype

```
void GFMmProcControlRecordStartRecord
(
    GFMMPROC_CONTROL_RATE handle
)
```

Parameters

handle	The handle of the record control object.

GFMmProcControlRecordStopRecord()

Stop recording the media. GFMmProcControlRecordStopRecord will not automatically stop the Player. It only stops the recording.

Stopping the Player does not imply a GFMmProcControlRecordStopRecord. Rather, the recording will be put into a "standby" mode. Once the Player is re-started, the recording will resume automatically. After GFMmProcControlRecordStopRecord, GFMmProcControlRecordStartRecord can be called to resume the recording. If GFMmProcControlRecordStopRecord is called when the recording has already stopped, it will be ignored. When GFMmProcControlRecordStopRecord returns, the recording has stopped and a RECORD_STOPPED event will be delivered through the player listener.

Function Prototype

```
void GFMmProcControlRecordStopRecord
(
    GFMMPROC_CONTROL_RATE handle
)
```

Parameters

handle	The handle of the record control object.

GFMmProcControlRecordReset()

Erase the current recording. If the recording is in progress, reset will implicitly call GFMmProcControlRecordStopRecord. Calling reset after commit will have no effect on the current recording. If the Player that is associated with this record control is closed, reset will be called implicitly.

Function Prototype

```
void GFMmProcControlRecordReset
(
    GFMMPROC_CONTROL_RATE handle
)
```

Parameters

handle	The handle of the record control object.

GFMMPROC_CONTROL_STOP_TIME Fields

GFMMPROC_CONTROL_STOP_TIME_RESET MAX_VALUE
Returned by GFMmProcControlStopTimeGetStopTime if no stop-time is set. Value NvS32 MAX_VALUE is assigned to GFMMPROC_CONTROL_STOP_TIME_RESET.

GFMmProcControlStopTimeGetStopTime()

Gets the last value successfully set by GFMmProcControlStopTimeGetStopTime. Returns the constant GFMMPROC_CONTROL_STOP_TIME_RESET if no stop time is set. This is the default.

Function Prototype

```
NvS32 GFMmProcControlStopTimeGetStopTime
(
    GFMMPROC_CONTROL_STOP_TIME handle
)
```

Parameters

`handle` The handle of the stop time control object.

Returns: The current stop time in microseconds.

GFMmProcControlStopTimeSetStopTime()

Sets the media time at which you want the player to stop. The player will stop when its media time reaches the stop-time. A GFMMPROC_PLAYER_LISTENER_STOPPED_AT-_TIME event will be delivered through the player listener.

The player is guaranteed to stop within one second past the preset stop-time (i.e. stop-time <= current-media-time <= stop-time + 1 sec.); unless the current media time is already passed the preset stop time when the stop time is set. If the current media time is already past the stop time set, the player will stop immediately. A GFMMPROC_PLAYER_LISTENER-_STOPPED_AT_TIME event will be delivered. After the player stops due to the stop-time set, the previously set stop-time will be cleared automatically. Alternatively, the stop

time can be explicitly removed by setting it to: GFMMPROC_CONTROL_STOP-_TIME_RESET.

You can always call GFMmProcControlStopTimeGetStopTime on a stopped player. To avoid a potential race condition, it is illegal to call GFMmProcControlStopTimeGetStopTime on a started player if a media stop-time has already been set.

Function Prototype

```
void GFMmProcControlStopTimeSetStopTime
(
    GFMMPROC_CONTROL_STOP_TIME handle,
    NvS32 stopTime
)
```

Parameters

handle	The handle of the stop time control object.
stopTime	The time in microseconds at which you want the player to stop, in media time.

GFMmProcControlTempoGetTempo()

Gets the current playback tempo. This represents the current state of the sequencer:

- A sequencer may not be initialized before the player is prefetched. An uninitialized sequencer in this case returns a default tempo of 120 beats per minute.
- After prefetching has finished, the tempo is set to the start tempo of the MIDI sequence (if any).
- During playback, the return value is the current tempo and varies with tempo events in the MIDI file.
- A stopped sequence retains the last tempo it had before it was stopped.
- A call to GFMmProcControlTempoSetTempo changes current tempo until a tempo event in the MIDI file is encountered.

Function Prototype

```
NvS32 GFMmProcControlTempoGetTempo
(
```

Function Prototype

```
    GFMMPROC_CONTROL_TEMPO handle
)
```

Parameters

`handle`	The handle of the tempo control object.

Returns: The current tempo, expressed in milli-beats per minute.

GFMmProcControlTempoSetTempo()

Sets the current playback tempo. Tempo is a volatile state of the sequencer. As MIDI sequences may contain META tempo events, tempo may change during playback of the sequence. Setting the tempo with GFMmProcControlTempoSetTempo does not prevent the tempo from being changed subsequently by tempo events in the MIDI sequence. Example: during playback of a sequence, the user changes the tempo. But just moments later, the MIDI sequence changes the tempo to another value, so effectively the user interaction is ignored. To overcome this, and to allow consistent user interaction, use GFMmProcControlRateSetRate inherited from rate control.

The GFMmProcControlTempoSetTempo method returns the actual tempo set by the player's implementation. It sets the tempo as close to the requested value as possible, but is not required to set it to the exact value. Specifically, implementations may have a lower or upper limit, which will be used as tempo if the requested tempo is out of limits. 0 or negative tempo does not exist and will always result in the lower tempo limit of the implementation. Implementations are guaranteed to support 10'000 to 300'000 milli-beats per minute.

Setting tempo to a stopped sequence will force the sequence to start with that tempo, even if the sequence has a tempo event at the start position. Any subsequent tempo events in the sequence will be considered, though. Rewinding back to a position with a tempo event will also result in a tempo change caused by the tempo event. Example: a sequence with initial tempo of 120bpm has not been started yet. The user sets the tempo to 140bpm and starts playback. When the playback position is then reset to the beginning, the tempo will be set to 120bpm due to the tempo event at the beginning of the sequence.

Playback rate (see GFMmProcControlRateSetRate) and tempo are independent factors of the effective tempo. Modifying tempo with GFMmProcControlTempoSetTempo does not affect the playback rate and vice versa. The effective tempo is the product of tempo and rate.

Function Prototype

```
NvS32 GFMmProcControlTempoSetTempo
(
    GFMMPROC_CONTROL_TEMPO handle,
    NvS32 millitempo
)
```

Parameters

`handle`	The handle of the tempo control object.
`millitempo`	The tempo specified in milli-beats per minute (must be > 0, e.g. 120'000 for 120 beats per minute).

Returns: The tempo that was actually set, expressed in milli-beats per minute.

GFMMPROC_CONTROL_TONE Fields

GFMMPROC_CONTROL_TONE_VERSION
The VERSION attribute tag.

GFMMPROC_CONTROL_TONE_TEMPO
The TEMPO event tag.

GFMMPROC_CONTROL_TONE_RESOLUTION
The RESOLUTION event tag.

GFMMPROC_CONTROL_TONE_BLOCK_START
Defines a starting point for a block.

GFMMPROC_CONTROL_TONE_BLOCK_END
Defines an ending point for a block.

GFMMPROC_CONTROL_TONE_PLAY_BLOCK
Play a defined block.

GFMMPROC_CONTROL_TONE_SET_VOLUME
The SET_VOLUME event tag.

GFMMPROC_CONTROL_TONE_REPEAT
The REPEAT event tag.

GFMMPROC_CONTROL_TONE_C4
Middle C.

GFMMPROC_CONTROL_TONE_SILENCE
Silence.

GFMmProcControlToneSetSequence()

Sets the tone sequence.

Function Prototype

```
void GFMmProcControlToneSetSequence
(
    GFMMPROC_CONTROL_TONE handle,
    char* sequence
)
```

Parameters

handle	The handle of the tone control object.
sequence	The sequence to set.

GFMmProcControlVolumeGetLevel()

Get the current volume level set. GFMmProcControlVolumeGetLevel may return -1 if and only if the player is in the REALIZED state (the audio device has not been initialized) and GFMmProcControlVolumeSetLevel has not yet been called.

Function Prototype

```
NvS32 GFMmProcControlVolumeGetLevel
(
    GFMMPROC_CONTROL_VOLUME handle
)
```

Parameters

handle	The handle of the volume control object.

Returns: The current volume level or -1.

GFMmProcControlVolumeIsMuted()

Get the mute state of the signal associated with this volume control.

Function Prototype

```
NvS32 GFMmProcControlVolumeIsMuted
(
    GFMMPROC_CONTROL_VOLUME handle
)
```

Parameters

handle	The handle of the volume control object.

Returns: The mute state

GFMmProcControlVolumeSetLevel()

Set the volume using a linear point scale with values between 0 and 100. 0 is silence; 100 is the loudest useful level that this volume control supports. If the given level is less than 0 or greater than 100, the level will be set to 0 or 100 respectively. When GFMmProcControlVolumeSetLevel results in a change in the volume level, a GFMMPROC_PLAYER-_LISTENER_VOLUME_CHANGED event will be delivered through the player listener.

Function Prototype

```
NvS32 GFMmProcControlVolumeSetLevel
(
    GFMMPROC_CONTROL_VOLUME handle,
    NvS32 level
)
```

Parameters

`handle`	The handle of the volume control object.
`level`	The new volume specified in the level scale.

Returns: The level that was actually set.

GFMmProcControlVolumeSetMute()

Mute or unmute the player associated with this volume control. Calling GFMmProcControlVolumeSetMute(1) on the player that is already muted is ignored, as is calling GFMmProcControlVolumeSetMute(0) on the player that is not currently muted. Setting mute on or off doesn't change the volume level returned by GFMmProcControlVolumeGetLevel. When GFMmProcControlVolumeSetMute results in a change in the muted state, a GFMMPROC_PLAYER_LISTENER_VOLUME_CHANGED event will be delivered through the player listener.

Function Prototype

```
void GFMmProcControlVolumeSetMute
(
    GFMMPROC_CONTROL_VOLUME handle,
    NvS32 mute
)
```

Parameters

`handle`	The handle of the volume control object.
`mute`	Specify non-zero to mute the signal, 0 to unmute the signal.

Advanced Processing

The next set of capabilities exposed for the NVIDIA multimedia processor further enhances the audio feature set and also provides a set of generic transforms for processing both audio and non-audio related data. (NOTE: At the present revision level of this programming interface, the advanced feature set requires that the basic audio feature set be included.) The main object at this level of functionality is the Global Manager. It is the access point for obtaining system dependent resources to provide the advanced audio functionality and generic transforms. The system dependent resources are encapsulated and accessed by a set of interfaces: effect module, sound source 3D, and media processor.

The effect module interface ties together a logic group of player objects that expose a common set of controls. This interface allows a parameter to be set on the interface of a control exposed by the effect module, and have its affect be applied to each player that supports the control tied to the effect module.

The sound source 3D interface is similar to the effect module interface; it also represents a logical group of player objects but exposes explicit controls to locate the connected sources in a virtual 3D acoustical environment. For both the effect module interface and the sound source 3D interface, the player is tied to the interface via a common, derived module interface. To enhance the capabilities of the virtual 3D environment, the global manager interface supports a method to retrieve the spectator interface. The spectator interface represents the listener in the virtual 3D environment. It exposes controls to move the listener relative to the other 3D sources in the virtual environment.

The media processor interface is designed to post-process different media types. The media processor object may generate asynchronous events that may be monitored by the client by implementing the media processor listener interface.

Global Manager

The global manager handles the creation of effect modules, sound source 3D controls and media processors. Furthermore, the spectator interface can also be retrieved from the global manager.

The global manager has similar methods to the controllable interface. It is supposed that implementations mostly return different subclasses of effect control. These effects will be treated as global effects concerning all players

Module

Module is an abstract base class for a logical group of players and/or MIDI channels. Typically, a module object as such can not be fetched from anywhere; the subclasses effect module and sound source 3D are used instead.

At least one player can always be added to a module, but adding a MIDI channel or a second player is not guaranteed to work. For instance, if the second player is of a type that would make some control already retrieved from the module become obsolete, then adding the second player is not allowed.

Effect Module

Effect module is a logical group of players to place effects on the players in the module. The term effect is used here to cover both effects extending effect control and effect-like controls extending controls like volume control. The global manager is used to retrieve effect modules.

See "Module" for interface details.

Sound Source 3D

Sound source 3D represents a sound source in a virtual acoustical space. It is a logical group of players and/or MIDI channels.

Typically, a directivity control, a distance attenuation control, a doppler control, a location control, an obstruction control and a macroscopic control can be fetched from the sound source 3D, if they are supported.

The global manager is used to retrieve Sound source 3Ds.

See "Module" for interface details.

Media Processor

Media processor is an interface designed to post-process different media types. It is intended that a media processor mostly exposes various effect controls and format controls.

The global manager is used to retrieve a media processor.

Media Processor Listener

Media processor listener is an interface that may be used to receive events generated by a media processor. To use this interface, instantiate an object that implements this interface and pass it to the GFMmProcMediaProcessorAddMediaProcessorListener method of the media processor.

- "GFMmProcMediaProcessorListenerMediaProcessorUpdate()" on page 315

Spectator

Spectator represents the listener in the virtual acoustical space.

Location control, orientation control and Doppler control, typically can be fetched from the spectator. They are used to control the listener model of the virtual acoustical space. Please read more about spatialization and virtual acoustics in location control.

Interface Details

See "Controllable" on page 250 for more details.

Advanced Processing Interface Details

GFMmProcGlobalManagerCreateEffectModule()

Creates an effect module.

Function Prototype

```
GFMMPROC_EFFECT_MODULE GFMmProcGlobalManagerCreateEffectModule
(
    GFMMPROC_GLOBAL_MANAGER handle
)
```

Parameters

`handle`	The handle of the global manager object.

Returns: An effect module object that may be used to group players.

GFMmProcGlobalManagerCreateMediaProcessor()

Creates a media processor object. Content type is passed as a MIME type. Format control and audio format control specify constants for a set of commonly used content types.

Function Prototype

```
GFMMPROC_MEDIA_PROCESSOR
GFMmProcGlobalManagerCreateMediaProcessor
(
    GFMMPROC_GLOBAL_MANAGER handle,
    char* contentType
)
```

Parameters

`handle`	The handle of the global manager object.
`contentType`	The content type of the source data to be processed.

Returns: An instance of media processor.

GFMmProcGlobalManagerCreateSoundSource3D()

Creates a sound source 3D object.

Function Prototype

```
GFMMPROC_SOUND_SOURCE_3D
GFMmProcGlobalManagerCreateSoundSource3D
(
    GFMMPROC_GLOBAL_MANAGER handle
)
```

Parameters

`handle`	The handle of the global manager object.

Returns: A sound source 3D object that represents a virtual sound source and that may be used to group players.

GFMmProcGlobalManagerGetControl()

Obtains the object that implements the specified control interface. If the specified control interface is not supported, then null is returned.

Function Prototype

```
GFMMPROC_CONTROL GFMmProcGlobalManagerGetControl
(
    GFMMPROC_GLOBAL_MANAGER handle,
    char* controlType
)
```

Parameters

`handle`	The handle of the global manager object.
`controlType`	The class name of the control.

GFMmProcGlobalManagerGetControls()

Obtains the collection of controls from the global manager. Since the global manager can implement multiple control interfaces, it is necessary to check each object against different control types. The list of control objects returned will not contain any duplicates and the list will not change over time. If no control is supported, a null is returned.

Function Prototype

```
GFMMPROC_CONTROL* GFMmProcGlobalManagerGetControls
(
    GFMMPROC_GLOBAL_MANAGER handle
)
```

Parameters

`handle`	The handle of the global manager object.

Returns: An array of control type objects.

GFMmProcGlobalManagerGetSpectator()

Gets the spectator, which represents the listener in the virtual acoustical space.

Function Prototype

```
GFMMPROC_SPECTATOR GFMmProcGlobalManagerGetSpectator
(
    GFMMPROC_GLOBAL_MANAGER handle
)
```

Parameters

`handle`	The handle of the global manager object.

Returns: The spectator interface.

GFMmProcGlobalManagergetSupportedMediaProcessorInputTypes()

Gets the supported media processor input content types.

Function Prototype

```
char*
GFMmProcGlobalManagergetSupportedMediaProcessorInputTypes
(
    GFMMPROC_GLOBAL_MANAGER handle
)
```

Parameters

`handle`	The handle of the global manager object.

Returns: A list of supported input content types.

GFMmProcGlobalManagerGetUnitsPerMeter()

Get the measures in use in units per meter. Initially, the value is 1000, that is, one integer unit corresponds to one millimeter in the physical world.

Function Prototype

```
NvS32 GFMmProcGlobalManagerGetUnitsPerMeter
(
    GFMMPROC_GLOBAL_MANAGER handle
)
```

Parameters

handle	The handle of the global manager object.

Returns: Units per meter.

GFMmProcGlobalManagerSetUnitsPerMeter()

Defines how many units one meter is. Initially, the value is 1000, that is, one integer unit to one millimeter in the physical world. Changing this value never moves any objects; it just defines what will be the unit used in various get and set methods (e.g. GFMmProcControlLocationSetCartesian) after calling this GFMmProcGlobalManagerSetUnitsPerMeter method.

Function Prototype

```
void GFMmProcGlobalManagerSetUnitsPerMeter
(
    GFMMPROC_GLOBAL_MANAGER handle
    NvS32 newScalingFactor
)
```

Parameters

handle	The handle of the global manager object.
newscalingFactor	

GFMmProcModuleAddMIDIChannel()

Adds an individual MIDI channel of a MIDI player to the module. If the played MIDI file or MIDI stream contains information that is contradictory to

what is specified via this module the behavior will be implementation specific.

Function Prototype

```
void GFMmProcModuleAddMIDIChannel
(
    GFMMPROC_MODULE handle,
    GFMMPROC_PLAYER player,
    NvS32 channel
)
```

Parameters

handle	The handle of the module object.
player	The MIDI player whose channel is to be added.
channel	The channel of the given player to be added. The range is 0..15.

GFMmProcModuleAddPlayer()

Adds a player to the module.

Function Prototype

```
void GFMmProcModuleAddPlayer
(
    GFMMPROC_MODULE handle,
    GFMMPROC_PLAYER player
)
```

Parameters

handle	The handle of the module object.
player	The player to be added.

GFMmProcModuleGetControl()

Obtain the object that implements the specified control interface. If the specified control interface is not supported then null is returned. If the controllable supports multiple objects that implement the same specified control interface, only one of them will be returned. To obtain all the control's

of that type, use the GFMmProcModuleGetControls method and check the list for the requested type.

Function Prototype

```
GFMMPROC_CONTROL GFMmProcModuleGetControl
(
    GFMMPROC_MODULE handle,
    char* controlType
)
```

Parameters

handle	The handle of the module object.
controlType	The class name of the control. The class name should be given either as the fully-qualified name of the class.

GFMmProcModuleGetControls()

Obtain the collection of controls from the object that implements this interface. Since a single object can implement multiple control interfaces, it's necessary to check each object against different control types.

Function Prototype

```
GFMMPROC_CONTROL* GFMmProcModuleGetControls
(
    GFMMPROC_MODULE handle
)
```

Parameters

handle	The handle of the module object.

GFMmProcModuleRemoveMIDIChannel()

Removes a MIDI channel from the module. All channels can be removed at once by GFMmProcModuleRemovePlayer.

Function Prototype

```
void GFMmProcModuleRemoveMIDIChannel
(
    GFMMPROC_MODULE handle,
    GFMMPROC_PLAYER player,
```

Function Prototype

```
    NvS32 channel
)
```

Parameters

handle	The handle of the module object.
player	The MIDI player whose channel is to be removed.
channel	The channel of the given player to be removed.

GFMmProcModuleRemovePlayer()

Removes a player or all channels of a player from the module.

Function Prototype

```
void GFMmProcModuleRemovePlayer
(
    GFMMPROC_MODULE handle,
    GFMMPROC_PLAYER player
)
```

Parameters

handle	The handle of the module object.
player	The player to be removed.

GFMMPROC_MEDIA_PROCESSOR Fields

GFMMPROC_MEDIA_PROCESSOR_UNKNOWN
Constant telling that either the length of the media or progress of the processing is unknown.

GFMmProcMediaProcessorAbort()

Aborts the processing even if the processing was not complete. Any bytes that were written to output may not be reasonable and should be discarded. A PROCESSING_COMPLETED event is posted and the media processor is

transitioned into SETUP state. The call is ignored if the media processor was already in SETUP state.

Function Prototype

```
void GFMmProcMediaProcessorAbort
(
    GFMMPROC_MEDIA_PROCESSOR handle
)
```

Parameters

handle	The handle of the media processor object.

GFMmProcMediaProcessorAddMediaProcessorListener()

Adds a media processor listener that will receive events generated by this media processor.

Function Prototype

```
void GFMmProcMediaProcessorAddMediaProcessorListener
(
    GFMMPROC_MEDIA_PROCESSOR handle,
    GFMMPROC_MEDIA_PROCESSOR_LISTENER mediaProcessorListener
)
```

Parameters

handle	The handle of the media processor object.
mediaProcessorListener	Add a media processor listener that will receive events generated by this media processor. If null, the request will be ignored.

GFMmProcMediaProcessorComplete()

Waits until the processing has been completed. If the media processor is not in STARTED state, calls start implicitly. Otherwise, waits until a

PROCESSING_COMPLETED event has been posted. After this method returns, the media processor is in SETUP state.

Function Prototype

```
void GFMmProcMediaProcessorComplete
(
    GFMMPROC_MEDIA_PROCESSOR handle
)
```

Parameters

handle	The handle of the media processor object.

GFMmProcMediaProcessorGetProgress()

Get an estimated percentage of work that has been done. If the media processor is in SETUP state the percentage of work that has been done is:

- 0, if the media processor has just been created, but no input has been set.
- 0, if new input has been set, but processing has not been started.
- Amount of work completed (0..100%), if processing has been started, stopped, completed or aborted.
- UNKNOWN, if the estimation cannot be calculated.

Function Prototype

```
NvS32 GFMmProcMediaProcessorGetProgress
(
    GFMMPROC_MEDIA_PROCESSOR handle
)
```

Parameters

handle	The handle of the media processor object.

Returns: The estimated percentage of completed work.

GFMmProcMediaProcessorremoveMediaProcessorLis-

tener()

Remove a media processor listener that was receiving events generated by this media processor.

Function Prototype

```
void GFMmProcMediaProcessorremoveMediaProcessorListener
(
    GFMMPROC_MEDIA_PROCESSOR handle,
    GFMMPROC_MEDIA_PLAYER_LISTENER mediaProcessorListener
)
```

Parameters

`handle`	The handle of the media processor object.
`mediaProcessorListener`	The listener to be removed. If the listener does not exist or is null, the request will be ignored.

GFMmProcMediaProcessorStop()

Stops processing temporarily. If the media processor is in a SETUP or STOPPED state, the call is ignored. Otherwise, moves to STOPPED state.

Function Prototype

```
void GFMmProcMediaProcessorStop
(
    GFMMPROC_MEDIA_PROCESSOR handle
)
```

Parameters

`handle`	The handle of the media processor object.

GFMmProcMediaProcessorstart()

Starts processing. If the media processor is in STARTED state, the call is ignored. Upon calling this method, the media processor moves to STARTED state. After the processing has been completed, a

PROCESSING_COMPLETED event will be delivered and the media processor will be transitioned into the SETUP state.

Function Prototype

```
void GFMmProcMediaProcessorstart
(
    GFMMPROC_MEDIA_PROCESSOR handle
)
```

Parameters

handle	The handle of the media processor object.

GFMmProcMediaProcessorsetOutput()

Sets the output of the media processor.

Function Prototype

```
void GFMmProcMediaProcessorsetOutput
(
    GFMMPROC_MEDIA_PROCESSOR handle,
    GFMMPROC_STREAM output
)
```

Parameters

handle	The handle of the media processor object.
output	The stream to be used as output.

GFMmProcMediaProcessorsetInput()

Sets the input of the media processor.

Function Prototype

```
void GFMmProcMediaProcessorsetInput
(
    GFMMPROC_MEDIA_PROCESSOR handle,
    GFMMPROC_STREAM input,
    NvS32 length
)
```

Parameters

`handle`	The handle of the media processor object.
`input`	The stream to be used as input.
`length`	The estimated length of the processed media in bytes. Since the input is given as an input stream, the implementation cannot find out what is the length of the media until it has been processed. The estimated length is used only when the progress method is used to query the progress of the processing. If the length is not known, UNKNOWN should be passed as a length.

GFMMPROC_MEDIA_PROCESSOR_LISTENER_PROCESSING Fields

GFMMPROC_MEDIA_PROCESSOR_LISTENER_PROCESSING_COMPLETED
Posted to indicate that the processing has been completed and that the media processor has transitioned into SETUP state. Event data is an integer that indicates whether the result was written successfully to the output stream or not.

GFMMPROC_MEDIA_PROCESSOR_LISTENER_PROCESSING_ERROR
Posted to indicate that the processing was interrupted because of some error and the media processor has transitioned into SETUP state. Event data is a string that indicates the error message.

GFMmProcMediaProcessorListenerMediaProcessorUpdate()

Called to deliver an event to a registered listener when a media processor event is observed.

Function Prototype

```
void GFMmProcMediaProcessorListenerMediaProcessorUpdate
(
    GFMMPROC_MEDIA_PROCESSOR_LISTENER handle,
    GFMMPROC_MEDIA_PROCESSOR processor,
    char* event,
    void* eventData
)
```

Parameters

`handle`	The handle of the media processor listener object.
`processor`	The media processor which generated the event.
`event`	The generated event.
`eventData`	The associated event data.

Advanced Controls

This package adds a number of controls for setting formats via Format Control and Audio Format Control, for defining the effect processing network structure via Effect Control and Effect Order Control, advanced MIDI channel specific controls through MIDI Channel Control, stereo panning through Pan Control, and moderating of Player priorities through Priority Control.

Format Control

Format control controls the format used for storing media. Formats are specified when media is captured from a player or if the format is changed when media is processed by a media processor. Format control is a super interface for audio format control. Format control specifies the methods to set and get various parameters to specify the media format.

Audio Format Control

Audio format control controls the setting of the audio format.

Effect Control

Effect control is an interface for controlling an abstract filter with various preset settings. Individual effects might have various parameters. The effects can be turned on and off by using a GFMmProcControlEffectSetEnabled method. By default, effects are disabled.

Effect Order Control

Effect order control is an interface designed to specify the order of effects represented by effect controls. It is intended that an effect order control might be exposed by Media processors and effect modules. Effects with a smaller order are processed first. If an effect order control is not used, the default ordering of effects will be used.

MIDI Channel Control

MIDI channel control is a control that gives access to MIDI-channel-specific controls. Essentially, it provides the same functionality as controllable, but per channel, not per player. A MIDI channel control might be supported for MIDI players. If the played MIDI file or MIDI stream contains information that is contradictory to what is specified via the MIDI channel control the behavior will be implementation specific.

Pan Control

Pan control is an interface for manipulating the panning of a player in the stereo output mix. If the input is stereo, this controls the balance between the channels.

Priority Control

Priority control is an interface for manipulating the priority of a player among other players.

Advanced Control Interface Details

GFMMPROC_CONTROL_FORMAT Fields

GFMMPROC_CONTROL_FORMAT_METADATA_NOT_SUPPORTED

Setting metadata is not supported.

GFMMPROC_CONTROL_FORMAT_METADATA_SUPPORTED_FIXED_KEYS
Setting metadata is supported and the set of metadata keys is limited.

GFMMPROC_CONTROL_FORMAT_METADATA_SUPPORTED_FREE_KEYS
Setting metadata is supported and any strings can be used as metadata keys.

GFMMPROC_CONTROL_FORMAT_PARAM_BITRATE
The output bit rate integer parameter.

GFMMPROC_CONTROL_FORMAT_PARAM_BITRATE
The bit rate type string parameter tells if the bit rate is constant or variable.

GFMMPROC_CONTROL_FORMAT_PARAM_FRAMERATE
The output frame rate integer parameter.

GFMMPROC_CONTROL_FORMAT_PARAM_QUALITY
The quality parameter for example for JPEG.

GFMMPROC_CONTROL_FORMAT_PARAM_SAMPLERATE
The output sample rate integer parameter.

GFMMPROC_CONTROL_FORMAT_PARAM_VERSION_TYPE
The format version type string parameter to specify what is the type of the format.

GFMmProcControlFormatGetEstimatedBitRate()

Gets the estimated bit rate of the media.

Function Prototype

```
NvS32 GFMmProcControlFormatGetEstimatedBitRate
(
    GFMMPROC_CONTROL_FORMAT handle
)
```

Parameters

`handle` The handle of the format control object.

Returns: An estimated bit rate in bits/second.

GFMmProcControlFormatGetFormat()

Get the format.

Function Prototype

```
char* GFMmProcControlFormatGetFormat
(
    GFMMPROC_CONTROL_FORMAT handle
)
```

Parameters

handle	The handle of the format control object.

Returns: The current format.

GFMmProcControlFormatGetIntParameterValue()

Gets the current value of an integer parameter.

Function Prototype

```
NvS32 GFMmProcControlFormatGetIntParameterValue
(
    GFMMPROC_CONTROL_FORMAT handle,
    char* parameter
)
```

Parameters

handle	The handle of the format control object.
parameter	The parameter to return.

Returns: The current value of the given parameter.

GFMmProcControlFormatGetMetadataOverride()

Gets the metadata override mode.

Function Prototype

```
NvS32 GFMmProcControlFormatGetMetadataOverride
(
    GFMMPROC_CONTROL_FORMAT handle
)
```

Parameters

handle	The handle of the format control object.

Returns: True if the metadata set by the application overrides the existing metadata, false if the existing metadata is preserved.

GFMmProcControlFormatGetMetadataSupportMode()

Returns the metadata support mode. The support mode is one of the following.

- GFMMPROC_CONTROL_FORMAT_METADATA_NOT_SUPPORTED
- GFMMPROC_CONTROL_FORMAT_METADATA_SUPPORTED_FIXED_KEYS
- GFMMPROC_CONTROL_FORMAT_METADATA_SUPPORTED_FREE_KEYS

Function Prototype

```
NvS32 GFMmProcControlFormatGetMetadataSupportMode
(
    GFMMPROC_CONTROL_FORMAT handle
)
```

Parameters

handle	The handle of the format control object.

Returns: The current metadata support mode.

GFMmProcControlFormatGetStrParameterValue()

Gets the current value of a string parameter.

Function Prototype

```
char* GFMmProcControlFormatGetStrParameterValue
(
    GFMMPROC_CONTROL_FORMAT handle,
    char* parameter
)
```

Parameters

handle	The handle of the format control object.
parameter	The parameter to return.

Returns: The current value of the given parameter.

GFMmProcControlFormatGetSupportedFormats()

Gets supported formats.

Function Prototype

```
char* GFMmProcControlFormatGetSupportedFormats
(
    GFMMPROC_CONTROL_FORMAT handle
)
```

Parameters

handle	The handle of the format control object.

Returns: An array of supported formats.

GFMmProcControlFormatGetSupportedIntParameter-Range()

Gets range for the given integer valued parameter.

Function Prototype

```
int* GFMmProcControlFormatGetSupportedIntParameterRange
(
```

Function Prototype

```
    GFMMPROC_CONTROL_FORMAT handle,
    char* parameter
)
```

Parameters

handle	The handle of the format control object.
parameter	The parameter to return.

Returns: Min and max value for the given parameter

GFMmProcControlFormatGetSupportedIntParameters()

Gets the supported parameters for the current format.

Function Prototype

```
char* GFMmProcControlFormatGetSupportedIntParameters
(
    GFMMPROC_CONTROL_FORMAT handle
)
```

Parameters

handle	The handle of the format control object.

Returns: An array of supported integer valued parameters

GFMmProcControlFormatGetSupportedMetadataKeys()

Returns a list of supported metadata keys. If any String can be used as a metadata GFMmProcControlFormatGetSupportedMetadataKeys returns a list of the most common metadata keys.

Function Prototype

```
char* GFMmProcControlFormatGetSupportedMetadataKeys
(
    GFMMPROC_CONTROL_FORMAT handle
)
```

Parameters

handle	The handle of the format control object.

Returns: The supported metadata keys or a list of the most common ones.

GFMmProcControlFormatGetSupportedStrParameters()

Gets the supported parameters for the current format.

Function Prototype

```
char* GFMmProcControlFormatGetSupportedStrParameters
(
    GFMMPROC_CONTROL_FORMAT handle
)
```

Parameters

handle	The handle of the format control object.

Returns: An array of supported string valued parameters.

GFMmProcControlFormatgetSupportedStrParameterValues()

Gets available values for the given string valued parameter.

Function Prototype

```
char* GFMmProcControlFormatgetSupportedStrParameterValues
(
    GFMMPROC_CONTROL_FORMAT handle,
    char* parameter
)
```

Parameters

handle	The handle of the format control object.
parameter	The parameter to return.

Returns: An array of supported values for the given parameter.

GFMmProcControlFormatSetFormat()

Sets the format.

Function Prototype

```
void GFMmProcControlFormatSetFormat
(
    GFMMPROC_CONTROL_FORMAT handle,
    char* format
)
```

Parameters

handle	The handle of the format control object.
format	

GFMmProcControlFormatSetMetadata()

Sets metadata for the media.

Function Prototype

```
void GFMmProcControlFormatSetMetadata
(
    GFMMPROC_CONTROL_FORMAT handle,
    char* key,
    char* value
)
```

Parameters

handle	The handle of the format control object.
key	The metadata key.
value	The metadata value.

GFMmProcControlFormatSetMetadataOverride()

Sets the metadata override mode. In the override mode, the metadata set by the application overrides the metadata existing in the media when the existing metadata is found out only when the processing of the media is already going on.

By default, the metadata override mode is true.

Function Prototype

```
void GFMmProcControlFormatSetMetadataOverride
(
    GFMMPROC_CONTROL_FORMAT handle,
    NvS32 override
)
```

Parameters

handle	The handle of the format control object.
override	True to make application set metadata to be written on top of the existing metadata, false to preserve the existing metadata.

GFMmProcControlFormatSetParameter()

Sets string valued parameter's value.

Function Prototype

```
void GFMmProcControlFormatSetParameter
(
    GFMMPROC_CONTROL_FORMAT handle,
    char* parameter,
    char* value
)
```

Parameters

handle	The handle of the format control object.
parameter	The parameter to set.
value	The string value of the parameter.

GFMmProcControlFormatSetParameter()

Sets integer valued parameter's value.

Function Prototype

```
NvS32 GFMmProcControlFormatSetParameter
(
    GFMMPROC_CONTROL_FORMAT handle,
```

Function Prototype

```
    char* parameter,
    NvS32 value
)
```

Parameters

handle	The handle of the format control object.
parameter	The parameter to set.
value	The string value of the parameter.

Returns: The value that was set.

Audio Formats

Audio format control controls the setting of the audio format. Audio format control serves two purposes:

- Set the audio codec in an audio-video format.
- Set the format for audio only content type. For example, a Player to play and record audio from a microphone provides an audio format control to specify the format.

For example, setting the recording format to 128 kbps MP3 is done as follows:

```
// Initializing the player and record control omitted
GFMMPROC_CONTROL_AUDIO_FORMAT format;
format = (GFMMPROC_CONTROL_AUDIO_FORMAT)
    GFMmProcPlayerGetControl(p,
GFMMPROC_CONTROL_AUDIO_FORMAT_STR);
GFMmProcControlFormatSetFormat(format, "audio/mpeg");
GFMmProcControlFormatSetParameter(format,
    GFMMPROC_CONTROL_FORMAT_PARAM_VERSION_TYPE,
"MPEG1_layer_3");
GFMmProcControlFormatSetParameter(format,
    GFMMPROC_CONTROL_FORMAT_PARAM_BITRATE, 128000);
```

Format Name	PARAM_VERSION_TYPE	Specification	Market Name

audio/mpeg	MPEG1_layer_2	MPEG-1 layer 2	MP2
audio/mpeg	MPEG1_layer_3	MPEG-1 layer 3	MP3
audio/mpeg	MPEG1_layer_3_pro	Coding technologies	MP3pro
audio/mpeg	MPEG2_AAC	MPEG-2 Advanced_Audio_Coding	AAC
audio/mpeg	MPEG4_HE_AAC	MPEG-4 High Efficiency Advanced Audio Coding	aacPlus
audio/3gpp	Enhanced_AAC_Plus	3GPP Enhanced aacPlus	Enhanced aacPlus
audio/amr	AMR	3GPP Adaptive Multi Rate	AMR
audio/amr-wb	AMR_WB	3GPP Wide Band Adaptive Multi Rate	AMR-WB
audio/amr-wb+	AMR_WB_Plus	3GPP Wide Band Adaptive Multi Rate plus	AMR-WB+
audio/x-gsm	GSM	ETSI GSM full-rate speech codec	GSM-FR
audio/x-gsmefr	GSM_EFR	ETSI GSM enhanced full-rate speech codec	GSM_EFR
audio/qcelp	QCELP	The Electronic Industries Association (EIA) & Telecommunications Industry Association (TIA) standard IS-733, "TR45: High Rate Speech Service Option for Wideband Spread Spectrum Communications Systems"	
audio/midi	MIDI	MIDI Manufacturers Association	MIDI
audio/spmidi	SP_MIDI	MIDI Manufacturers Association	SP-MIDI
audio/x-wav	MS_PCM	Microsoft, linear Pulse Coded Modulation	MS-WAV
audio/x-wav	MS_ADPCM	Microsoft, Adaptive Delta Pulse Coded Modulation	MS-ADPCM

audio/x-wav	YAMAHA_ADPCM	Yamaha, Adaptive Delta Pulse Coded Modulation	Yamaha, ADPCM
audio/basic	AU		uLaw
audio/x-vorbis	OGG_VORBIS	OGG Vorbis	OGG Vorbis
audio/x-realaudio	REALAUDIO_8	RealAudio ver 8	RealAudio
audio/x-aiff	AIFF	AIFF uncompressed	AIFF
audio/x-ms-wma	WMA_9	Windows Media Audio	WMA

GFMMPROC_CONTROL_EFFECT Fields

GFMMPROC_CONTROL_EFFECT_SCOPE_LIVE_ONLY
A constant used to identify the live part of the effect queue. For example, the live part of the effect chain might be the audio played out a speaker.

GFMMPROC_CONTROL_EFFECT_SCOPE_RECORD_ONLY
A constant used to identify the recording part of the effect queue. For example, in the case of audio, the live effects are applied to the speaker and the record effects affect only the audio recorded from a microphone.

GFMMPROC_CONTROL_EFFECT_SCOPE_LIVE_AND_RECORD
A constant used to identify both parts of the effect queue. For example, in the case of audio, the effects are applied both for the speaker and and recorded audio (e.g. acoustic echo cancellation).

GFMmProcControlEffectGetPreset()

Gets the current preset.

Function Prototype

```
char* GFMmProcControlEffectGetPreset
(
```

Function Prototype

```
    GFMMPROC_CONTROL_EFFECT handle
)
```

Parameters

handle	The handle of the effect control object.

Returns: The preset that is set at the moment. If none of the presets is set, null will be returned.

GFMmProcControlEffectGetPresetNames()

Gets the available preset names.

Function Prototype

```
char* GFMmProcControlEffectGetPresetNames
(
    GFMMPROC_CONTROL_EFFECT handle
)
```

Parameters

handle	The handle of the effect control object.

Returns: The names of all the available preset modes.

GFMmProcControlEffectGetScope()

Returns the scope in which the effect is present.

Function Prototype

```
NvS32 GFMmProcControlEffectGetScope
(
    GFMMPROC_CONTROL_EFFECT handle
)
```

Parameters

handle	The handle of the effect control object.

Returns: `SCOPE_LIVE_ONLY`, `SCOPE_RECORD_ONLY` or `SCOPE_LIVE_AND_RECORD`

GFMmProcControlEffectIsEnabled()

Returns true if the effect is enabled and false otherwise.

Function Prototype

```
NvS32 GFMmProcControlEffectIsEnabled
(
    GFMMPROC_CONTROL_EFFECT handle
)
```

Parameters

handle	The handle of the effect control object.

Returns: non-zero = enabled, 0 = disabled

GFMmProcControlEffectIsEnforced()

Returns the current enforced setting of the effect.

Function Prototype

```
NvS32 GFMmProcControlEffectIsEnforced
(
    GFMMPROC_CONTROL_EFFECT handle
)
```

Parameters

handle	The handle of the effect control object.

Returns: Non-zero if the effect is an enforced effect, 0 if not.

GFMmProcControlEffectSetEnabled()

Enables/disables the effect.

Function Prototype

```
void GFMmProcControlEffectSetEnabled
(
    GFMMPROC_CONTROL_EFFECT handle,
    NvS32 enable
)
```

Parameters

handle	The handle of the effect control object.
enable	non-zero = enable, 0 = disable

GFMmProcControlEffectSetEnforced()

Enforces the effect to be in use.

If this is an effect control of a media processor, the enforced setting does not affect in any way.

Function Prototype

```
void GFMmProcControlEffectSetEnforced
(
    GFMMPROC_CONTROL_EFFECT handle,
    NvS32 enforced
)
```

Parameters

handle	The handle of the effect control object.
enforced	True if the effect is essential and cannot be dropped, false if the effect can be dropped if the system runs out of resources.

GFMmProcControlEffectSetPreset()

Sets the effect according to the given preset.

Function Prototype

```
void GFMmProcControlEffectSetPreset
(
    GFMMPROC_CONTROL_EFFECT handle,
    char* preset
)
```

Parameters

handle	The handle of the effect control object.
preset	The new preset that will be taken into use.

GFMmProcControlEffectSetScope()

Sets the scope of the effect. If this is an effect control of the media processor, the scope setting does not affect in anything.

Function Prototype

```
void GFMmProcControlEffectSetScope
(
    GFMMPROC_CONTROL_EFFECT handle,
    NvS32 scope
)
```

Parameters

handle	The handle of the effect control object.
scope	GFMMPROC_CONTROL_EFFECT_SCOPE_LIVE_ONLY, GFMMPROC_CONTROL_EFFECT_SCOPE_RECORD_ONLY or GFMMPROC_CONTROL_EFFECT_SCOPE_LIVE_AND_RECORD.

GFMmProcControlEffectOrderGetEffectOrder()

Returns the current position of the effect in the effect processing chain.

Function Prototype

```
NvS32 GFMmProcControlEffectOrderGetEffectOrder
(
    GFMMPROC_CONTROL_EFFECT_ORDER handle,
    GFMMPROC_CONTROL_EFFECT effect
)
```

Parameters

handle	The handle of the effect order control object.
effect	The effect whose order is queried.

Returns: A value ranged to the whole Integer range, smaller values meaning earlier effects.

GFMmProcControlEffectOrderGetEffectOrders()

Returns the current positions of the effects in the effect processing chain.

Function Prototype

```
GFMMPROC_CONTROL_EFFECT
GFMmProcControlEffectOrderGetEffectOrders
(
    GFMMPROC_CONTROL_EFFECT_ORDER handle
)
```

Parameters

handle	The handle of the effect order control object.

Returns: An array of EffectControls in their current order where the first item in the array means the effect that is processed first.

GFMmProcControlEffectOrderSetEffectOrder()

Sets the order of the effect in the effect chain. If the implementation does not allow a certain ordering of the effects, it may reorder the effects as it sees fit.

Function Prototype

```
NvS32 GFMmProcControlEffectOrderSetEffectOrder
(
    GFMMPROC_CONTROL_EFFECT_ORDER handle,
    GFMMPROC_CONTROL_EFFECT effect,
    NvS32 order
)
```

Parameters

handle	The handle of the effect order control object.
effect	The effect whose order is to be set.
order	The desired position of the effect in the effect chain. The range is the whole Integer range.

Returns: The actual order of the effect that was set.

GFMmProcControlMidiChannelGetChannelControl()

Obtains the object that implements the specified control interface for the given channel. If the specified control interface is not supported then null is returned.

Function Prototype

```
GFMMPROC_CONTROL GFMmProcControlMidiChannelGetChannelControl
(
    GFMMPROC_CONTROL_MIDI_CHANNEL handle,
    char* controlType,
    NvS32 channel
)
```

Parameters

handle	The handle of the MIDI channel control object.
controlType	The class name of the control. The class name should be given either as the fully-qualified name of the class.
channel	Number of the channel. It must be in the range 0..15.

Returns: The object that implements the control, or null.

GFMmProcControlMidiChannelGetChannelControls()

Obtains the collection of controls for the given channel. Since a single channel can implement multiple Control interfaces, it is necessary to check each object against different control types. The list of control objects returned will not contain any duplicates and the list will not change over time. If no control is supported, a zero is returned.

Function Prototype

```
GFMMPROC_CONTROL* GFMmProcControlMidiChannelGetChannelControls
(
    GFMMPROC_CONTROL_MIDI_CHANNEL handle,
    NvS32 channel
)
```

Parameters

handle	The handle of the MIDI channel control object.
channel	Number of the channel. It must be in the range 0..15.

Returns: The collection of control objects.

GFMmProcControlPanGetPan()

Gets the current panning set.

Function Prototype

```
NvS32 GFMmProcControlPanGetPan
(
    GFMMPROC_CONTROL_PAN handle
)
```

Parameters

handle	The handle of the pan control object.

Returns: The current panning.

GFMmProcControlPanSetPan()

Sets the panning using a linear point scale with values between -100 and 100. 0 represents panning for both channels, -100 full panning to the left and 100 full panning to the right. If the given panning value is less than -100 or greater than 100, the panning will be set to -100 or 100, respectively.

Function Prototype

```
NvS32 GFMmProcControlPanSetPan
(
    GFMMPROC_CONTROL_PAN handle,
    NvS32 pan
)
```

Parameters

handle	The handle of the pan control object.
pan	The new panning to be set.

Returns: The panning that was actually set.

GFMmProcControlPriorityGetPriority()

Gets the current priority.

Function Prototype

```
NvS32 GFMmProcControlPriorityGetPriority
(
    GFMMPROC_CONTROL_PRIORITY handle
)
```

Parameters

handle	The handle of the priority control object.

Returns: The current priority.

GFMmProcControlPrioritySetPriority()

Sets the new priority.

Function Prototype

```
Void GFMmProcControlPrioritySetPriority
(
    GFMMPROC_CONTROL_PRIORITY handle,
    NvS32 priority
)
```

Parameters

handle	The handle of the priority control object.
priority	

Advanced 3D Audio Controls

This package adds a number of controls for localizing the audio sounds in a virtual acoustical environment. The Commit Control interface provides a mechanism to enable many audio parameters to be updated simultaneously. The Orientation Control is an interface for manipulating the virtual orientation of an object in the virtual acoustical space. The Directivity Control interface adds to Orientation Control a method for setting the directivity pattern of a sound source. The Distance Attenuation Control is an interface for controlling how the sound from a sound source is attenuated with its distance from the listener. The Doppler Control is an interface for manipulating the settings of an effect called *Doppler*. The Location Control is an interface for manipulating the virtual location of an object (usually a sound source 3D or the listener via the spectator) in the virtual acoustic space. The macroscopic control is an interface for manipulating the macroscopic behavior of a sound source when using 3D audio. The obstruction control provides a mechanism to control the overall level of an audio signal flowing directly from a sound source to the listener.

Commit Control

Commit control provides a mechanism to enable many audio parameters to be updated simultaneously. In many ways, it can be considered to be the analog of a 'Draw' or 'Flush' method of a graphics system. The commit control, if it is supported, can only be obtained from the global manager.

- "GFMmProcControlCommitCommit()" on page 341
- "GFMmProcControlCommitIsDeferred()" on page 342
- "GFMmProcControlCommitSetDeferred()" on page 342

Orientation Control

Orientation control is an interface for manipulating the virtual orientation of an object in the virtual acoustical space. Usually, the object is either the listener via the Spectator or a source via a sound source 3D. In the case of a sound source, subinterfaces directivity control and/or macroscopic control are used instead of the superinterface orientation control.

- "GFMmProcControlOrientationGetOrientationVectors()" on page 343
- "GFMmProcControlOrientationSetOrientationXYZ()" on page 343

Directivity Control

Directivity control adds to orientation control a method for setting the directivity pattern of a sound source.

Distance Attentuation Control

DistanceAttenuationControl is an interface for controlling how the sound from a sound source is attenuated with its distance from the listener. If the control is supported, it can be retrieved from a sound source 3D.

Doppler Control

Doppler control is an interface for manipulating the settings of an effect called Doppler. It can be obtained either for the spectator (in which case it specifies the listener's velocity) or a sound source 3D (in which cases it specifies a sound source's velocity). The Doppler control must be enabled before it has any effect. Please see the section below on Synthesizing Doppler with pitch control if Doppler control is not supported.

Location Control

Location control is an interface for manipulating the virtual location of an object (usually a sound source 3D or the listener via the spectator) in the virtual acoustic space.

Macroscopic Control

Macroscopic control is an interface for manipulating the macroscopic behavior of a sound source when using 3D audio. By default sound sources act as point sources having a zero size. Macroscopic control enables one to specify a finite size for a sound source. This is useful for relatively big sound sources like waterfalls. This control can be fetched from a sound source 3D if it is supported at all.

Obstruction Control

Obstruction control provides a mechanism to control the overall level of an audio signal flowing directly from a sound source to the listener. Additionally, it provides the capability of filtering (attenuating) the high frequency components of the signal to a specified degree.

3D Audio Controls Interface Details

GFMmProcControlCommitCommit()

Transfers all the pending parameters to the audio processing system.

In the immediate mode, a call to this method is ignored.

Function Prototype

```
void GFMmProcControlCommitCommit
(
    GFMMPROC_CONTROL_COMMIT handle
)
```

Parameters

handle	The handle of the commit control object.

GFMmProcControlCommitIsDeferred()

Returns the mode of the commit control.

Function Prototype

```
NvS32 GFMmProcControlCommitIsDeferred
(
    GFMMPROC_CONTROL_COMMIT handle
)
```

Parameters

handle	The handle of the commit control object.

Returns: Non-zero if setting of properties is deferred.

GFMmProcControlCommitSetDeferred()

Sets the mode of the commit control. When switching back from the deferred mode to the immediate mode by calling GFMmProcControlCommitSetDeferred(0), all the pending parameters from the buffer are transmitted to the audio processing system automatically.

Function Prototype

```
void GFMmProcControlCommitSetDeferred
(
    GFMMPROC_CONTROL_COMMIT handle,
    NvS32 deferred
)
```

Parameters

handle	The handle of the commit control object.
deferred	A non-zero integer indicates the property setting should be deferred.

GFMmProcControlOrientationGetOrientationVectors()

Gets the orientation of the object using two vectors.

Function Prototype

```
int* GFMmProcControlOrientationGetOrientationVectors
(
    GFMMPROC_CONTROL_ORIENTATION handle
)
```

Parameters

handle	The handle of the orientation control object.

Returns: A 6-element array that has first the front vector and then the up vector.

GFMmProcControlOrientationSetOrientationXYZ()

Turns the object to the new orientation.

The new orientation is given using rotation angles. A zero vector corresponds to the orientation pointing directly towards the negative Z-axis. Orientation is applied in the following order: heading, pitch, and roll. Therefore, notice that heading turns the X-axis and therefore affects the pitch, and similarly heading and pitch turn the Z-axis and therefore affect the roll.

Function Prototype

```
void GFMmProcControlOrientationSetOrientationXYZ
(
    GFMMPROC_CONTROL_ORIENTATION handle,
    NvS32 heading,
    NvS32 pitch,
    NvS32 roll
)
```

Parameters

handle	The handle of the orientation control object.
heading	The rotation around the Y-axis in degrees.
pitch	The rotation around the X-axis in degrees.
roll	The rotation around the Z-axis in degrees.

GFMmProcControlOrientationSetOrientationVectors()

Turns the object to the new orientation.

The orientation is specified using two vectors, one specifying the direction of the front vector of the object in world coordinates, and another specifying the "above" vector of the object. The right and up vectors of the object are calculated by first normalizing both source vectors, then calculating the right vector as the cross product of the "above" vector and the front vector, and the up vector as a cross product of the front and right vectors. Because both vectors are normalized, they may be of any length.

Function Prototype

```
void GFMmProcControlOrientationSetOrientationVectors
(
    GFMMPROC_CONTROL_ORIENTATION handle,
    int* frontVector,
    int* aboveVector
)
```

Parameters

handle	The handle of the orientation control object.
frontVector	A 3-element array specifying the front vector of the object in the world coordinate system.
above Vector	A 3-element array specifying the "above" vector mentioned above.

GFMmProcControlDirectivityGetParameters()

Gets the directivity pattern of a sound source.

Function Prototype

```
int* GFMmProcControlDirectivityGetParameters
(
```

Function Prototype

```
    GFMMPROC_CONTROL_DIRECTIVITY handle
)
```

Parameters

`handle`	The handle of the directivity control object.

Returns: An array of type int[3] containing the minAngle, maxAngle and rearLevel parameters, in that order.

GFMmProcControlDirectivitySetParameters()

Sets all the directivity parameters simultaneously.

The minAngle and maxAngle parameters define the inner and outer sound cones, respectively. The symmetry axis of the cones is the orientation axis defined by GFMmProcControlOrientationSetOrientationXYZ/Vectors. The effect of the cones is as follows:

- Within the inner cone, the sound from a source is not attenuated due to directivity and is therefore at its loudest.
- Between the inner and outer cones, the level of the perceived sound decreases linearly as the angle of the listener from the axis of orientation of the sound source increases.
- Outside of the outer cone, the sound is heard at a level rearLevel relative to the level inside the inner cone.

The sound can be completely muted in the region outside the outer cone by setting rearLevel to MIN_VALUE. (In practice, the implementation will probably then interpolate linearly in the logarithmic (level) domain from the maximum level (at minAngle) to the quietest level it supports (at maxAngle), which will be approximately -90 dB if the native processing uses 16-bit samples, or about -140 dB for 24-bit samples.)

Setting the rearLevel to 0 or setting both angles to 180 degrees causes the sound source to revert once again to being an omnidirectional source (the default state: no directivity attenuation).

Function Prototype

```
void GFMmProcControlDirectivitySetParameters
(
    GFMMPROC_CONTROL_DIRECTIVITY handle,
```

Function Prototype

```
    NvS32 minAngle,
    NvS32 maxAngle,
    NvS32 rearLevel
)
```

Parameters

handle	The handle of the directivity control object.
minAngle	An angle in degrees (a value from 0 to 180 inclusive), measured from the axis of orientation, defining the inner cone, within which the sound from a source is not attenuated due to directivity.
maxAngle	An angle in degrees (a value from 0 to 180 inclusive, where minAngle <= maxAngle), defining the outer cone.
rearLevel	the level in millibels (mB, 1 mB = 1/100 dB), at which the sound from a source will be heard at maxAngle and wider angles; must be a non-positive value. Setting the rearLevel to 0 makes the source omnidirectional.

GFMmProcControlDistanceAttenuationGetMaxDistance()

Returns the maximum distance. At the maximum distance, the gain does not decrease any more. The exact behavior of the gain at distances beyond the maximum distance depends on the value of muteAfterMax.

Function Prototype

```
NvS32 GFMmProcControlDistanceAttenuationGetMaxDistance
(
    GFMMPROC_CONTROL_DISTANCE_ATTENUATION handle
)
```

Parameters

handle	The handle of the distance attenuation control object.

Returns: The maximum distance, specified in units defined by GFMmProcGlobalManagerGetUnitsPerMeter.

GFMmProcControlDistanceAttenuationGetMinDistance()

Returns the distance where the source is loudest. The gain does not increase if the distance gets smaller than this.

Function Prototype

```
NvS32 GFMmProcControlDistanceAttenuationGetMinDistance
(
    GFMMPROC_CONTROL_DISTANCE_ATTENUATION handle
)
```

Parameters

`handle`	The handle of the distance attenuation control object.

Returns: The minimum distance, specified in units defined by GFMmProcGlobalManagerGetUnitsPerMeter.

GFMmProcControlDistanceAttenuationGetMuteAfter-Max()

Returns how the distance gain behaves for distances beyond the maximum distance.

Function Prototype

```
NvS32 GFMmProcControlDistanceAttenuationGetMuteAfterMax
(
    GFMMPROC_CONTROL_DISTANCE_ATTENUATION handle
)
```

Parameters

`handle`	The handle of the distance attenuation control object.

Returns: Non-zero if beyond the maximum distance the source is silent, or 0 if beyond the maximum distance the source's gain is held constant at the level at the maximum distance.

GFMmProcControlDistanceAttenuationGetRolloffFac-

tor()

Returns the rolloff factor for the distance gain.

Function Prototype

```
NvS32 GFMmProcControlDistanceAttenuationGetRolloffFactor
(
    GFMMPROC_CONTROL_DISTANCE_ATTENUATION handle
)
```

Parameters

handle	The handle of the distance attenuation control object.

Returns: The rolloff factor as an exponent specified in thousandths.

GFMmProcControlDistanceAttenuationSetParameters()

Sets all the 3D audio distance attenuation parameters simultaneously. Distances are specified in units defined by GFMmProcGlobalManagerGetUnitsPerMeter.

Function Prototype

```
void GFMmProcControlDistanceAttenuationSetParameters
(
    GFMMPROC_CONTROL_DISTANCE_ATTENUATION handle,
    NvS32 minDistance,
    NvS32 maxDistance,
    NvS32 muteAfterMax,
    NvS32 rolloffFactor
)
```

Parameters

handle	The handle of the distance attenuation control object.
minDistance	The minimum distance, below which the distance gain is clipped to its maximum value of 1.0.
maxDistance	The maximum distance, beyond which the distance gain does not decrease any more. The exact behaviour of the gain at distances beyond the maximum distance depends on the value of the muteAfterMax.

Parameters

`muteAfterMax`	An integer determining how the distance gain behaves at distances greater than maxDistance: non-zero if beyond the maximum distance the source is silent; 0 if beyond the maximum distance the source's gain is held constant at the level at the maximum distance.
`rolloffFactor`	The rolloff factor, specified in thousandths (1000 representing a rolloff factor of 1.0, 2000 representing 2.0 and 500 representing 0.5). Higher values cause the distance gain to attenuate more quickly.

GFMmProcControlDopplerGetVelocityCartesian()

Returns the current velocity, used in calculations for the Doppler effect. The velocity is specified using right-handed Cartesian components in units defined by GFMmProcGlobalManagerGetUnitsPerMeter.

Function Prototype

```
int* GFMmProcControlDopplerGetVelocityCartesian
(
    GFMMPROC_CONTROL_DOPPLER handle
)
```

Parameters

`handle`	The handle of the Doppler control object.

Returns: An integer array containing the x, y and z components of the current velocity or a zero-length array if the velocity is unknown.

GFMmProcControlDopplerIsEnabled()

Returns whether this Doppler effect is currently active.

Function Prototype

```
NvS32 GFMmProcControlDopplerIsEnabled
(
    GFMMPROC_CONTROL_DOPPLER handle
)
```

Parameters

handle	The handle of the Doppler control object.

Returns: Non-zero if Doppler is being applied.

lGFMmProcControlDopplerSetEnabled()

Specifies if this Doppler effect is active or ignored. In case the Doppler control is fetched from the spectator, this method does not affect anything.

Function Prototype

```
void GFMmProcControlDopplerSetEnabled
(
    GFMMPROC_CONTROL_DOPPLER handle,
    NvS32 dopplerEnabled
)
```

Parameters

handle	The handle of the Doppler control object.
doppleEnabled	A non-zero integer specifying if this Doppler effect is to be applied.

GFMmProcControlDopplerSetVelocityCartesian()

Sets the velocity, used in calculations for the Doppler effect. The velocity is specified using right-handed Cartesian components in units defined by GFMmProcGlobalManagerGetUnitsPerMeter. For example, if the x parameter is specified to be 5000 and GFMmProcGlobalManagerGetUnitsPerMeter returns 1000 the actual x component will be 5 m/s. The same applies to y and z parameters.

Function Prototype

```
void GFMmProcControlDopplerSetVelocityCartesian
(
    GFMMPROC_CONTROL_DOPPLER handle,
    NvS32 x,
    NvS32 y,
    NvS32 z
)
```

Parameters

handle	The handle of the Doppler control object.
x	The x component of the new velocity.
y	The y component of the new velocity.
z	The z component of the new velocity

GFMmProcControlDopplerSetVelocitySpherical()

Sets the velocity, used in calculations for the Doppler effect. The velocity is specified using spherical components. The radius component is specified in units defined by GFMmProcGlobalManagerGetUnitsPerMeter. For example, if the radius parameter is specified to be 5000 and GFMmProcGlobalManagerGetUnitsPerMeter returns 1000 the actual radius component will be 5 m/s.

Function Prototype

```
void GFMmProcControlDopplerSetVelocitySpherical
(
    GFMMPROC_CONTROL_DOPPLER handle,
    NvS32 azimuth,
    NvS32 elevation,
    NvS32 radius
)
```

Parameters

handle	The handle of the Doppler control object.
azimuth	The azimuth angle of the new velocity in degrees.
elevation	The elevation angle of the new velocity in degrees.
radius	The magnitude of the new velocity (namely the speed).

GFMmProcControlLocationGetCartesian()

Gets the coordinates of the current location. The measures are defined in units specified by GFMmProcGlobalManagerGetUnitsPerMeter.

Function Prototype

```
int* GFMmProcControlLocationGetCartesian
(
```

Function Prototype

```
    GFMMPROC_CONTROL_LOCATION handle
)
```

Parameters

handle	The handle of the location control object.

Returns: The x, y and z coordinates of the current location.

GFMmProcControlLocationSetCartesian()

Moves the object to the new location. Sets the location using Cartesian right-handed coordinates that are relative to the origin. The measures are defined in units specified by GFMmProcGlobalManagerGetUnitsPerMeter.

Function Prototype

```
void GFMmProcControlLocationSetCartesian
(
    GFMMPROC_CONTROL_LOCATION handle,
    NvS32 x,
    NvS32 y,
    NvS32 z
)
```

Parameters

handle	The handle of the location control object.
x	The x-coordinate of the new location.
y	The y-coordinate of the new location.
z	The z-coordinate of the new location.

GFMmProcControlLocationSetSpherical()

Moves the object to the new location.

Sets the new location using spherical coordinates. The negative z-axis is the reference. That is, a location where both azimuth and elevation are zero, is on

the negative z-axis. Radius is defined in units specified by GFMmProcGlobalManagerGetUnitsPerMeter.

Function Prototype

```
void GFMmProcControlLocationSetSpherical
(
    GFMMPROC_CONTROL_LOCATION handle,
    NvS32 azimuth,
    NvS32 elevation,
    NvS32 radius
)
```

Parameters

handle	The handle of the location control object.
azimuth	The azimuth angle of the new location in degrees. The azimuth is measured from the negative z-axis in the direction of the x-axis.
elevation	The elevation angle of the new location in degrees. The elevation is measured from the x-z-plane in the direction of the y-axis.
radius	The radius of the new location from the origin.

GFMmProcControlMacroscopicGetSize()

Gets the current size.

Function Prototype

```
int* GFMmProcControlMacroscopicGetSize
(
    GFMMPROC_CONTROL_MACROSCOPIC handle
)
```

Parameters

handle	The handle of the macroscopic control object.

GFMmProcControlMacroscopicSetSize()

Sets the size of the audio source. Setting all the lengths to zero disables the macroscopic behavior, causing the object to revert once again to being a point source (the default state). All the lengths are specified in units defined by

GFMmProcGlobalManagerGetUnitsPerMeter . The center of the macroscopic object stays at the point specified with location control.

Note: The lengths are specified in the object's own (rotated) axes, as defined via the orientation control by the last call to GFMmProcControlOrientationSetOrientationXYZ /Vectors.

Function Prototype

```
void GFMmProcControlMacroscopicSetSize
(
    GFMMPROC_CONTROL_MACROSCOPIC handle,
    NvS32 x,
    NvS32 y,
    NvS32 z
)
```

Parameters

handle	The handle of the macroscopic control object.
x	The "width" of the sound source in its transformed X (or "right") dimension.
y	The "height" of the sound source in its transformed Y (or "up") dimension.
z	The "thickness" or "depth" of the sound source in its transformed Z (or "front") dimension.

GFMmProcControlObstructionGetHFLevel()

Returns the level at which high frequency components are being transmitted directly from the source.

Function Prototype

```
NvS32 GFMmProcControlObstructionGetHFLevel
(
    GFMMPROC_CONTROL_OBSTRUCTION handle
)
```

Parameters

handle	The handle of the obstruction control object.

Returns: The level at which 5000 Hz frequency components are being transmitted in millibels.

GFMmProcControlObstructionGetLevel()

Returns the overall level being transmitted directly from the source.

Function Prototype

```
NvS32 GFMmProcControlObstructionGetLevel
(
    GFMMPROC_CONTROL_OBSTRUCTION handle
)
```

Parameters

handle	The handle of the obstruction control object.

Returns: The overall level being transmitted in millibels.

GFMmProcControlObstructionSetHFLevel()

Sets the level at which high frequency components will be transmitted directly from the source.

Function Prototype

```
void GFMmProcControlObstructionSetHFLevel
(
    GFMMPROC_CONTROL_OBSTRUCTION handle,
    NvS32 HFLevel
)
```

Parameters

`handle`	The handle of the obstruction control object.
`HFLevel`	The new output level at 5000 Hz in millibels (mB, 1 mB = 1/100 dB) at which the sound directly from a source will be heard compared to the overall level of the source; must be a non-positive value. Setting the level to 0 makes the source's frequency response flat.

GFMmProcControlObstructionSetLevel()

Sets the overall level that will be transmitted directly from the source.

Function Prototype

```
void GFMmProcControlObstructionSetLevel
(
    GFMMPROC_CONTROL_OBSTRUCTION handle,
    NvS32 level
)
```

Parameters

`handle`	The handle of the obstruction control object.
`level`	The new output level in millibels (mB, 1 mB = 1/100 dB) at which the sound directly from a source will be heard compared to a non-obstructed source; must be a non-positive value. Setting the level to 0 makes the source non-obstructed.

Advanced Audio Effect Controls

This package adds effect controls for audio. The audio virtualizer control interface virtualizes audio channels. The acoustic echo cancellation control interface provides a mechanism to remove content being played from content being recorded. The chorus control is an interface for manipulating the settings of an audio effect called chorus and its special case flanger. The equalizer control interface is an extension of the audio effect control interface for manipulating the equalization settings of a player(s). The reverb control is an interface for manipulating the settings of an audio effect called reverb. The reverb source control is an interface for manipulating the feeding from an object to the audio effect called reverb.

Audio Virtualizer Control

Audio virtualizer control is an effect to virtualize audio channels. The exact behavior of this effect is dependent on the number of audio channels the player has and the type of audio output channels of the device. For example, in the case of a stereo player and stereo headphone output, a stereo widening is used when this effect is used. A player with 5.1-channels uses similarly some virtual surround algorithm.

Interface Details

See "Effect Control" on page 317 for API details.

Acoustic Echo Cancellation Control

Acoustic echo cancellation control is an effect to remove content being played from content being recorded.

Interface Details

See "Effect Control" on page 317 for API details.

Chorus Control

Chorus control is an interface for manipulating the settings of an audio effect called chorus and its special case flanger.

- "GFMmProcControlEffectChorusGetAverageDelay()" on page 359
- "GFMmProcControlEffectChorusGetMaxAverageDelay()" on page 359

Equalizer Control

Equalizer control is an audio effect control for manipulating the equalization settings of a player(s). Equalizers (EQ) are usually used for two reasons: to compensate an unideal frequency response of a system to make it sound more natural or to create intentionally some unnatural coloring to the sound to create an effect. The equalizer in this API serves both of these purposes.

Reverb Control

Reverb control is an interface for manipulating the settings of an audio effect called reverb. A reverb control can only be fetched from the global manager and/or media processor (if reverb control is supported at all).

Reverb Source Control

Reverb source control is an interface for manipulating the audio signal from an object to the audio reverb effect. A reverb source control can only be fetched from an effect module or a sound source 3D (the terminal nodes of the effects network, if reverb source control is supported at all).

Audio Effect Control Interface Details

GFMmProcControlEffectChorusGetAverageDelay()

Gets the average delay.

Function Prototype

```
NvS32 GFMmProcControlEffectChorusGetAverageDelay
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle
)
```

Parameters

`handle`	The handle of the chorus control object.

Returns: The current average delay in microseconds.

GFMmProcControlEffectChorusGetMaxAverageDelay()

Gets the maximum supported average delay.

Function Prototype

```
NvS32 GFMmProcControlEffectChorusGetMaxAverageDelay
(
```

Function Prototype

```
    GFMMPROC_CONTROL_EFFECT_CHORUS handle
)
```

Parameters

handle	The handle of the chorus control object.

Returns: The maximum supported average delay in microseconds.

GFMmProcControlEffectChorusGetMaxModulation-Depth()

Gets the maximum supported delay modulation depth.

Function Prototype

```
NvS32 GFMmProcControlEffectChorusGetMaxModulationDepth
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle
)
```

Parameters

handle	The handle of the chorus control object.

Returns: The maximum supported delay modulation depth in microseconds.

GFMmProcControlEffectChorusGetMaxModulationRate()

Gets the maximum supported delay modulation rate.

Function Prototype

```
NvS32 GFMmProcControlEffectChorusGetMaxModulationRate
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle
)
```

Parameters

handle	The handle of the chorus control object.

Returns: The maximum supported delay modulation rate in mHz.

GFMmProcControlEffectChorusGetMinModulationRate()

Gets the minimum supported delay modulation rate.

Function Prototype

```
NvS32 GFMmProcControlEffectChorusGetMinModulationRate
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle
)
```

Parameters

`handle`	The handle of the chorus control object.

Returns: The minimum supported delay modulation rate in mHz.

GFMmProcControlEffectChorusGetModulationDepth()

Gets the current delay modulation depth.

Function Prototype

```
NvS32 GFMmProcControlEffectChorusGetModulationDepth
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle
)
```

Parameters

`handle`	The handle of the chorus control object.

Returns: The current delay modulation depth in microseconds.

GFMmProcControlEffectChorusGetModulationRate()

Gets the delay modulation rate.

Function Prototype

```
NvS32 GFMmProcControlEffectChorusGetModulationRate
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle
)
```

Parameters

handle	The handle of the chorus control object.

Returns: The current delay modulation rate in mHz.

GFMmProcControlEffectChorusGetWetLevel()

Gets the effect's wet level.

Function Prototype

```
NvS32 GFMmProcControlEffectChorusGetWetLevel
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle
)
```

Parameters

handle	The handle of the chorus control object.

Returns: The effect wet level for the effect in percents.

GFMmProcControlEffectChorusSetAverageDelay()

Sets the average delay. A typical average delay for a chorus would be 20000 microseconds and for a flanger 3000 microseconds.

Function Prototype

```
void GFMmProcControlEffectChorusSetAverageDelay
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle,
    NvS32 delay
)
```

Parameters

handle	The handle of the chorus control object.
delay	The new average delay in microseconds.

GFMmProcControlEffectChorusSetModulationDepth()

Sets the delay modulation peak-to-peak depth. A typical depth for a chorus would be 16000 microseconds and for a flanger 5000 microseconds.

Function Prototype

```
void GFMmProcControlEffectChorusSetModulationDepth
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle,
    NvS32 depth
)
```

Parameters

handle	The handle of the chorus control object.
depth	The new delay modulation peak-to-peak depth in microseconds.

GFMmProcControlEffectChorusSetModulationRate()

Sets the delay modulation peak-to-peak depth. A typical depth for a chorus would be 16000 microseconds and for a flanger 5000 microseconds.

Function Prototype

```
void GFMmProcControlEffectChorusSetModulationRate
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle,
    NvS32 rate
)
```

Parameters

handle	The handle of the chorus control object.
rate	The new delay modulation peak-to-peak depth in microseconds.

GFMmProcControlEffectChorusSetWetLevel()

Sets the effect's wet level. The effect wet level affects how much of the media is passed via the effect in percents. 100 means that all the media is processed in the effect and 0 means that no processing is done at all and all the media is bypassed. Values between 0 and 100 affect the wet/dry ratio of the processing

in the accuracy that the system supports. The effect is considered to be "wetter" when the effect wet level rises.

Function Prototype

```
void GFMmProcControlEffectChorusSetWetLevel
(
    GFMMPROC_CONTROL_EFFECT_CHORUS handle,
    NvS32 level
)
```

Parameters

handle	The handle of the chorus control object.
level	The new effect wet level for the effect in percents.

Returns: The value that was actually set.

GFMMPROC_CONTROL_EFFECT_EQUALIZER Fields

GFMMPROC_CONTROL_EFFECT_EQUALIZER_UNDEFINED
The undefined value.

GFMmProcControlEffectEqualizerGetBand()

Gets the band that has the most effect on the given frequency. If no band has effect on the given frequency, -1 is returned.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerGetBand
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle,
    NvS32 frequency
)
```

Parameters

handle	The handle of the equalizer control object.
frequency	The frequency in milliHertz which is to be equalized via the returned band.

Returns: The frequency band that has most effect on the given frequency or -1 if no band has effect on the given frequency.

GFMmProcControlEffectEqualizerGetBandLevel()

Gets the gain set for the given equalizer band.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerGetBandLevel
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle,
    NvS32 band
)
```

Parameters

handle	The handle of the equalizer control object.
band	The frequency band whose gain is asked. The numbering of the bands starts from 0 and ends at (GFMmProcControlEffectEqualizerGetNumberOfBands(handle) - 1).

Returns: The gain set for the given band in millibels.

GFMmProcControlEffectEqualizerGetBass()

Gets the bass level.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerGetBass
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle
)
```

Parameters

handle	The handle of the equalizer control object.

Returns: The current level that is set to the bass band. **GFMMPROC_CONTROL_EFFECT_EQUALIZER_UNDEFINED** is returned if the bass level is not defined

GFMmProcControlEffectEqualizerGetCenterFreq()

Gets the center frequency of the given band.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerGetCenterFreq
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle,
    NvS32 band
)
```

Parameters

handle	The handle of the equalizer control object.
band	The frequency band whose center frequency is asked. The numbering of the bands starts from 0 and ends at (GFMmProcControlEffectEqualizerGetNumberOfBands(handle) - 1).

Returns: The center frequency in milli-Hertz.

GFMmProcControlEffectEqualizerGetMaxBandLevel()

Returns the maximum band level supported.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerGetMaxBandLevel
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle
)
```

Parameters

handle	The handle of the equalizer control object.

Returns: The maximum band level in millibels.

GFMmProcControlEffectEqualizerGetMinBandLevel()

Returns the minimum band level supported.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerGetMinBandLevel
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle
)
```

Parameters

`handle`	The handle of the equalizer control object.

Returns: The minimum band level in millibels.

GFMmProcControlEffectEqualizerGetNumberOfBands()

Gets the number of frequency bands that the equalizer supports.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerGetNumberOfBands
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle
)
```

Parameters

`handle`	The handle of the equalizer control object.

Returns: The number of frequency bands that the equalizer supports.

GFMmProcControlEffectEqualizerGetTreble()

Gets the treble level.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerGetTreble
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle
)
```

Parameters

handle	The handle of the equalizer control object.

Returns: The current level that is set to the treble band. If the treble level cannot been defined EqualizerControl.UNDEFINED will be returned.

GFMmProcControlEffectEqualizerSetBandLevel()

Sets the given equalizer band to the given gain value.

Function Prototype

```
void GFMmProcControlEffectEqualizerSetBandLevel
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle,
    NvS32 level,
    NvS32 band
)
```

Parameters

handle	The handle of the equalizer control object.
level	The new gain in millibels that will be set to the given band. GFMmProcControlEffectEqualizerGetMinBandLevel and GFMmProcControlEffectEqualizer GetMaxBandLevel will define the maximum and minimum values.
band	The frequency band that will have the new gain. The numbering of the bands starts from 0 and ends at (GFMmProcControlEffectEqualizerGetNumberOfBands(handle) - 1).

GFMmProcControlEffectEqualizerSetBass()

Sets the bass level using a linear point scale with values between 0 and 100:

- a value of 0 applies the maximum available attenuation to frequencies in the bass band;
- a value of 50 gives a flat equalization of the bass band; and
- a value of 100 applies the maximum available amplification to frequencies in the bass band.

All the previous settings will be lost on the bass band.

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerSetBass
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle,
    NvS32 level
)
```

Parameters

handle	The handle of the equalizer control object.
level	The new level on a linear point scale that will be set to the bass band.

Returns: The level that was actually set.

GFMmProcControlEffectEqualizerSetTreble()

Sets the treble level using a linear point scale with values between 0 and 100:

- a value of 0 applies the maximum available attenuation to frequencies in the treble band;
- a value of 50 gives a flat
- equalization of the treble band; and
- a value of 100 applies the maximum available amplification to frequencies in the treble band.

All the previous settings will be lost on the treble band. Returns:

Function Prototype

```
NvS32 GFMmProcControlEffectEqualizerSetTreble
(
    GFMMPROC_CONTROL_EFFECT_EQUALIZER handle,
    NvS32 level
)
```

Parameters

handle	The handle of the equalizer control object.
level	The new level on a linear point scale that will be set to the treble band.

The level that was actually set.

GFMmProcControlEffectReverbGetReverbLevel()

Gets the gain level of the reverberation.

Function Prototype

```
NvS32 GFMmProcControlEffectReverbGetReverbLevel
(
    GFMMPROC_CONTROL_EFFECT_REVERB handle
)
```

Parameters

handle	The handle of the reverb control object.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Returns: The level of the reverberation in millibels.

GFMmProcControlEffectReverbGetReverbTime()

Gets the reverberation time, as set either explicitly via GFMmProcControlEffectReverbSetReverbTime or implicitly via GFMmProcControlEffectSetPreset (whichever was called last). If neither of these methods has been called, GFMmProcControlEffectReverbGetReverbTime returns the default reverberation time (that corresponding to "smallroom").

Function Prototype

```
NvS32 GFMmProcControlEffectReverbGetReverbTime
(
    GFMMPROC_CONTROL_EFFECT_REVERB handle
)
```

Parameters

handle	The handle of the reverb control object.

Returns: The current reverberation time in milliseconds.

GFMmProcControlEffectReverbSetReverbLevel()

Sets the gain level of the reverberation. The value is the relative level of the first reflections compared to the sound source without possible distance attenuations, directivities or obstructions taken into account.

Function Prototype

```
NvS32 GFMmProcControlEffectReverbSetReverbLevel
(
    GFMMPROC_CONTROL_EFFECT_REVERB handle,
    NvS32 level
)
```

Parameters

handle	The handle of the reverb control object.
level	The new level of the reverberation in millibels.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Returns: The value that was actually set.

GFMmProcControlEffectReverbSetReverbTime()

Sets the reverberation time of the reverb. The reverberation time is the time taken for the reverberant sound to attenuate by 60 dB from its initial level. Typical values are in the range from 100 to 20000 milliseconds.

The implementation might not support long reverberation times. Therefore, the actual time used might be shorter than the time specified with this method.

Function Prototype

```
void GFMmProcControlEffectReverbSetReverbTime
(
    GFMMPROC_CONTROL_EFFECT_REVERB handle,
```

Function Prototype

```
    NvS32 time
)
```

Parameters

handle	The handle of the reverb control object.
time	New reverberation time in milliseconds.

GFMmProcControlReverbSourceGetRoomLevel()

Gets the object specific level for the reverberant sound.

Function Prototype

```
NvS32 GFMmProcControlReverbSourceGetRoomLevel
(
    GFMMPROC_CONTROL_REVERB_SOURCE handle
)
```

Parameters

handle	The handle of the reverb source control object.

Returns: The current level of the reflected sound compared to the natural room level in millibels or DISCONNECT.

GFMMPROC_CONTROL_REVERB_SOURCE Fields

GFMMPROC_CONTROL_REVERB_SOURCE_DISCONNECT
The application can use this for GFMmProcControlReverbSourceSetRoomLevel when it does not intend to connect this object (typically effect module or Sound source 3D) to the reverb. Setting this value will disconnect the object from the global reverb. Also, the implementation can use this to optimize the performance and totally remove feeds into the reverb that never will be used. The method GFMmProcControlReverbSourceSetRoomLevel will not be changed if the application tries to change this in a non-supported state. Typically, it may be supported to use the DISCONNECT in the REALIZED state but not in any other state of the involved players.

GFMmProcControlReverbSourceSetRoomLevel()

Sets the object specific level for the reverberant sound.

Function Prototype

```
void GFMmProcControlReverbSourceSetRoomLevel
(
    GFMMPROC_CONTROL_REVERB_SOURCE handle,
    NvS32 level
)
```

Parameters

handle	The handle of the reverb source control object.
level	The level of the reflected sound compared to the natural room gain in millibels (mB, 1 mB = 1/100 dB); must be a non-positive value. The default value is 0 meaning the natural room gain (set by reverb control's presets). Setting this value as integer MIN_VALUE disables the reflected sound for the given object. With the value DISCONNECT, the object can be disconnected from the reverb.

CHAPTER 10

JPEG Encoder API (GFJxEncAPI)

GFJxEncAPI Reference

The JPEG encoder API (GFJxEncAPI) is an abstraction of the JPEG encoder hardware. The hardware JPEG encoder can accept data from different sources—a video input port, host YUV FIFO, display (GC), StretchBlt, and a JPEG decoder memory row. The encoder supports only baseline sequential JPEG encoding using standard Huffman tables for YUV4:2:0 and YUV4:2:2. The YUV4:2:2 encoding format is supported on the GoForce 4800 and later GPUs. Quantization tables are programmable, and the application can set them using the **`GFJxSetAttribute()`** function. The hardware JPEG encoder generates only a bit stream. The API attaches a JFIF header by default. The application can suppress header generation if it wants to generate its own header.

Digital zoom is implemented in the API in two passes. In the first pass, a cropped portion of the image is encoded and kept in GPU memory. In the second pass, encoded bits are decoded and routed to the encoder through a video scalar. The encoded data is pulled out by the host CPU.

The API also supports re-encoding the JPEG bit stream. In the re-encoding case, the API invokes the JPEG decoder and routes the decoded data to the encoder. Re-encoding can be used for various purposes, such as reducing the

size of a picture, shrinking the image resolution by scaling or cropping, and limiting the size of the image using rate control.

The hardware encoder also supports partial rotation of JPEG data, which then should be processed by software to generate a fully compliant bit stream for the rotated image. This feature is recommended for rotating large images from the camera. On some GPUs, full rotation is also available for a limited frame size.

The GFJxEncAPI consists of the functions described under "GFJxEncAPI Functions" on page 376, and the data structures described under "GFJxEncAPI Data Structures" on page 391.

GFJxEncAPI Functions

The GFJxEncAPI functions include the following:

GFJxEncGetProperty()

This function returns information, including the version of the GFJxEncAPI module. It is a good practice to call this function to query for the API version and its capabilities before using the rest of the GFJxEncAPI functions.

Function Prototype

```
GF_RETTYPE  GFJxEncGetProperty(
    GF_HANDLE    JXHandle,
    PGFPROPERTY  pJXProp );
```

Parameters

JXHandle	Handle specific to the GFJxEncAPI.
pJXProp	Pointer to GFJxEncAPI encoder properties. See "GFPROPERTY" on page 392.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxEncSetAttribute()

Sets a feature of the JPEG encoder hardware. Most of the time an application should only set a quantization table for luminance or chrominance. These quantization tables are in the same format as standard quantization tables (1 to 255). Other features are hardware dependent.

Function Prototype

```
GF_RETTYPE  GFJxEncSetAttribute(
    GF_HANDLE  JXEncHandle,
    NvU32      uiFeature,
    NvU32      pInfo );
```

Parameters

JXEncHandle	Handle specific to the GFJxEncAPI.
uiFeature	See "GFJxEncSetAttribute() Definitions" on page 377.
*pInfo	Pointer to the information buffer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxEncSetAttribute() Definitions

```
/*
   Definitions for uiFeature.
   The following definitions are for sets only.
*/
#define JX_ENC_SET_100QUALITY                          0
    // Preset QTable: items in luma and chroma Qtable are 1.
```

GFJxEncSetAttribute() Definitions (continued)

```
#define JX_ENC_SET_85QUALITY                              1
    // Preset QTable: standard Qtable A and B, divide by 2.
#define JX_ENC_SET_75QUALITY                              2
    // Preset QTable: standard Qtable A and B
#define JX_ENC_SET_50QUALITY                              3
    // Preset QTable: standard Qtable C and D
#define JX_ENC_SET_25QUALITY                              4
    // Preset QTable: standard Qtable C and D, mult. by 2.
#define JX_ENC_SET_12QUALITY                              5
    // Preset QTable: standard Qtable C and D, mult. by 4.
/*
    The following definitions can be used for sets and gets.
*/
#define JX_ENC_LQTABLE                                    6
    // Custom QTable: luma Qtable is 64x16 entries
#define JX_ENC_CQTABLE                                    7
    // Custom QTable: chroma Qtable is 64x16 entries
#define JX_ENC_FILTER                                     8
    // Decimation filter enable/disable.
    // 1 in *pInfo means enable.
    // 0 in *pInfo means disable.
#define JX_ENC_L_OFF_GAIN                                 9
    // Set luma offset and gain
#define JX_ENC_C_OFF_GAIN                                10
    // Set chroma offset and gain
#define JX_ENC_MAX_LBLOCK                                11
    // Maximum bits in luma block
#define JX_ENC_MAX_CBLOCK                                12
    // Maximum bits in chroma block
#define JX_ENC_HUFF_BSTUFF                               13
    // Huffman bit stuffing.
    // 1 in *pInfo means API does stuffing.
    // 0 in *pInfo means API does not do stuffing.
#define JX_ENC_MOTION_JPEG                               14
    // 1 in *pInfo enables motion JPEG capturing mode.
    // 0 in *pInfo disables motion JPEG capturing mode;
    //   use JPEG mode.
#define JX_ENC_RAW_DATA                                  15
    // 1 in *pInfo enables raw capture mode.
    // 0 in *pInfo enables the encode capture mode.
```

GFJxEncSetAttribute() Definitions (continued)

```
#define JX_ENC_BITSTREAM                                16
    // Re-encode the bit stream.
    // *pInfo is a pointer to PGFJXENCCALLBACK.
#define JX_ENC_ENCODE_FORMAT                            17
    // *pInfo gets the GFJxEncAPI encoding image color
    // format configuration.
#define JX_ENC_MAX_SIZE                                 18
    // *pInfo is the maximum byte count. GFJxEncAPI does
    // not generate a JPEG bit stream larger than this.
#define JX_ENC_FOCAL_POINT                              19
    // Takes effect only when JX_ENC_MAX_SIZE is set.
    // pInfo[0] is the focal point X position.
    // pInfo[1] is the focal point Y position.
#define JX_ENC_DEGRADATION                              20
    // pInfo[0] is the X direction degradation percentage.
    // pInfo[1] is the Y direction degradation percentage.
    // JX_ENC_DEGRADATION takes effect when JX_ENC_MAX_SIZE
    // is set. The focal point gets more bits in the bit
    // stream, which means it is the clearest point. A
    // position farther from the focal point gets fewer
    // bits allocated to it. The number of bits is linearly
    // degraded based on the percentage of degradation in
    // the X and Y directions.
#define JX_ENC_ROTATION                                 21
    // Gets the GFJxEncAPI rotation option, which is
    // specified in *pInfo. It can assume one of the four
    // rotation values in combination with JX_ENC_H_FLIP
    // and JX_ENCV_VFLIP.
#define JX_ENC_PARTIAL_ROTATE                           22
    // When the partial rotation option is set before
    // GFJxEncStart(), the hardware encoder partially
    // rotates the encoded data according to the rotation
    // option. The encoded data fetched from the encoder
    // after setting the partial rotation option cannot be
    // decoded by a normal decoder. It has to be processed
    // by calling GFJxEncPRComplete() to be converted into
    // a standard JPEG bit stream.
    // JX_ENC_PARTIAL_ROTATE takes the same parameter in
    // *pInfo as in JX_ENC_ROTATION. An image that is not a
    // multiple of the minimum coded unit (MCU) may not
    // produce a proper bit stream with partial rotation.
```

GFJxEncSetAttribute() Definitions (continued)

```
#define JX_ENC_ROTATE_0                                  0x0
    // Rotate to 0 degrees.
#define JX_ENC_ROTATE_90                                 0x1
    // Rotate to 90 degrees.
#define JX_ENC_ROTATE_180                                0x2
    // Rotate to 180 degrees.
#define JX_ENC_ROTATE_270                                0x3
    // Rotate to 270 degrees.
/*
    Horizontal flip, vertical flip, and one of the four
    rotations can be combined.
*/
#define JX_ENC_H_FLIP                                   0x10
    // Horizontal flip.
#define JX_ENC_V_FLIP                                   0x20
    // Vertical flip.
#define JX_ENC_ROTATE_UNSET                              0xff
    /** Option for attribute #JX_ENC_PARTIAL_ROTATE: Reset and
        disable partial rotation.
        Pass this to GFJxEncSetAttribute() for attribute
        #JX_ENC_PARTIAL_ROTATE to unset the Partial Rotation
        Flag and reset the PR Engine.
    */


#define JX_ENC_BYPASS_FIFO                               23
    /** Interpretation of associated data block:
        NvU32 Raw data capturing mode enable
          0 = Fetching encoded data through DMA FIFO
          1 = Fetching encoded data directly through output
              buffer not through DMA FIFO
     */
```

GFJxEncGetAttribute()

Gets a feature of the JPEG encoder hardware.

Function Prototype

```
GF_RETTYPE  GFJxEncGetAttribute(
    GF_HANDLE  JXEncHandle,
```

Function Prototype (continued)

```
    NvU32       uiFeature,
    NvU32       *pInfo );
```

Parameters

JXEncHandle	Handle specific to the GFJxEncAPI.
uiFeature	See "GFJxEncGetAttribute() Definitions".
*pInfo	Pointer to the information buffer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxEncGetAttribute() Definitions

```
/*
    Definitions for uiFeature
*/
#define JX_ENC_LQTABLE                                  6
    // Custom QTable: luma Qtable is 64x16 entries
#define JX_ENC_CQTABLE                                  7
    // Custom QTable: chroma Qtable is 64x16 entries
#define JX_ENC_FILTER                                   8
    // Decimation filter enable/disable.
    // 1 in *pInfo means enable.
    // 0 in *pInfo means disable.
#define JX_ENC_L_OFF_GAIN                               9
    // Set luma offset and gain
#define JX_ENC_C_OFF_GAIN                              10
    // Set chroma offset and gain
#define JX_ENC_MAX_LBLOCK                              11
    // Max Bits in luma block
#define JX_ENC_MAX_CBLOCK                              12
    // Max bits in chroma block
#define JX_ENC_HUFF_BSTUFF                             13
    // Huffman bit stuffing.
    // 1 in *pInfo means API does stuffing.
    // 0 in *pInfo means API does not do stuffing.
#define JX_ENC_MOTION_JPEG                             14
    // 1 enables motion JPEG capturing mode
    // 0 disables motion JPEG capturing mode
```

GFJxEncGetAttribute() Definitions

```
#define JX_ENC_RAW_DATA                                 15
    // 1 in *pInfo enables raw capture mode.
    // 0 in *pInfo enables the encode capture mode.
#define JX_ENC_ENCODE_FORMAT                            17
    // The GFJxEncAPI supports YUV420 and YUV422 encoding.
    // This attribute in *pInfo tells the GFJxEncAPI to
    // encode an image as the specified format. The
    // GFJxEncAPI converts the incoming image to either
    // YUV422 or YUV420 format, and then encodes it in the
    // JPEG bit stream.
    // See "GFRMSURFACE Color Format Definitions" on page 38.
#define JX_ENC_MAX_SIZE                                 18
    // *pInfo is the maximum byte count. The GFJxEncAPI
    // will not generate a JPEG bit stream larger than this.
#define JX_ENC_FOCAL_POINT                              19
    // Takes effect only when JX_ENC_MAX_SIZE is set.
    // pInfo[0] is the focal point X position.
    // pInfo[1] is the focal point Y position.
#define JX_ENC_DEGRADATION                              20
    // pInfo[0] is the X direction degradation percentage.
    // pInfo[1] is the Y direction degradation percentage.
    // JX_ENC_DEGRADATION takes effect when JX_ENC_MAX_SIZE
    // is set. The focal point gets more bits in the bit
    // stream, which means it is the clearest point. A
    // position farther from the focal point gets fewer
    // bits allocated to it. The number of bits is linearly
    // degraded based on percentage of degradation in the
    // X and Y directions.
#define JX_ENC_ROTATION                                 21
    // Gets the GFJxEncAPI to capture a rotated image.
    // *pInfo specifies the rotation option.
#define JX_ENC_PARTIAL_ROTATE                           22
    // Returns current setting of partial rotation.



#define JX_ENC_ROTATE_0                                0x0
    // Rotate to 0 degrees.
#define JX_ENC_ROTATE_90                               0x1
    // Rotate to 90 degrees.
#define JX_ENC_ROTATE_180                              0x2
    // Rotate to 180 degrees.
```

GFJxEncGetAttribute() Definitions

```
#define JX_ENC_ROTATE_270                                0x3
    // Rotate to 270 degrees.
#define JX_ENC_H_FLIP                                   0x10
    // Horizontal flip.
#define JX_ENC_V_FLIP                                   0x20
    // Vertical flip.



#define JX_ENC_BYPASS_FIFO                               23
    /** Interpretation of associated data block:
        NvU32 Raw data capturing mode enable
          0 = Fetching encoded data through DMA FIFO
          1 = Fetching encoded data directly through output
              buffer not through DMA FIFO
     */
#define JX_ENC_PIECE_MEAL                                24
    /** Interpretation of associated data block:
        NvU32 Piece-Meal encoding enable
          0 = enable
          1 = disable
     */
```

GFJxEncSetupInterrupt()

The application could use this function to set up the interrupt callback function. This function must be called before **`GFJxEncCapture()`**.

If this function returns **`GF_SUCCESS`**, whenever encoding is triggered, the GFJXEncAPI calls the interrupt callback function. Inside of the callback function, the application can call **`GFJxEncFetchImage()`** to fetch the encoded image.

If this function returns **`GF_ERROR`**, the application should use a polling scheme.

Function Prototype

```
GF_RETTYPE  GFJxEncSetupInterrupt(
    GF_HANDLE  JXEncHandle,
    void       (*Inter)(void *),
    void        *IPara );
```

Parameters

JXEncHandle	Handle specific to the GFJxEncAPI.
*Inter	Pointer to encoder application's interrupt callback function .
*IPara	Pointer to encoder application's interrupt callback function parameter.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error, check pError.

GFJxEncCapture()

Triggers the JPEG encoder for a frame and also displays the same frame in a preview window in decimated form. If the callback interrupt is set up, use the callback function to fetch data, otherwise, use a polling scheme.

This function can also copy the captured data into buffers that the application can provide. If **nBuf** is 0, the function returns right away and status is updated to **JXENC_NEW_FRAME** to indicate data is pending.

If **nBuf** is not zero, it contains the number of buffers in the array. The application must fill all the fields in the buffer structure and must make sure **nBytesCaptured** is set to 0. The **nBytesCaptured** field is updated by the GFJXEncAPI to show the actual number of bytes captured. The GFJXEncAPI keeps filling the buffers in the array until it exhausts all the buffers. If all the buffers are exhausted and there is more data to be copied, the function returns **JXENC_MORE_FETCH**. The application should then keep calling **GFJxEncFetchImage()** to fetch the remaining data.

Each buffer in the array should be a multiple of 64 bytes for better performance.

Function Prototype

```
GF_RETTYPE  GFJxEncCapture(
    GF_HANDLE     JXEncHandle,
    PGFJXENC_BUF  aBuf[],
    NvU32         nBuf,
    NvU32         *pStatus );
```

Parameters

JXEncHandle	Handle specific to the GFJxEncAPI.
aBuf[]	Pointer to "GFJXENC_BUF" on page 394.
nBuf	Number of buffers in array aBuf[].
*pStatus	Encoded data status.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxEncFetchImage()

Captures the JPEG encoded bit stream in the given system buffer. Capturing the encoded image can be done in polling or interrupt mode. It should be called after **GFJxEncCapture()**.

Function Prototype

```
GF_RETTYPE  GFJxEncFetchImage(
    GF_HANDLE     JXEncHandle,
    PGFJXENC_BUF  aBuf[],
    NvU32         nBuf,
    NvU32         *pStatus );
```

Parameters

JXEncHandle	Handle specific to the GFJxEncAPI.
aBuf[]	Pointer to "GFJXENC_BUF" on page 394.
nBuf	Number of buffers in array aBuf[].
*pStatus	Encoded data status.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxEncStart()

GFJxEncStart() is called just before real encoding begins. It sets up the hardware encoder for the encoding to be done by a subsequent call to

GFJxEncCapture(). See "Encoding Options" and "Piecemeal Encoding from the Host" below.

After calling **GFJxEncStart()**, an application should call **GFJxEncCapture()** to trigger the encoder and, optionally, to fetch the encoded data. If data is not fetched in **GFJxEncCapture()**, the application must call **GFJxEncFetchImageData()** to fetch the encoded or raw data. The application should continue calling it until all the encoded or raw data is processed.

An application can fetch as many frames as desired after calling the **GFJxEncStart()** function.

Function Prototype

```
GF_RETTYPE  GFJxEncStart(
    GF_HANDLE      JXEncHandle,
    PGFJXENCSTART  pStart );
```

Parameters

JXEncHandle	Handle specific to the GFJxEncAPI.
pStart	Pointer to "GFJXENCSTART" on page 392.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

Encoding Options

Encoding can be done from the following sources:

- Video input port (VIP, camera, cropping, and decimation are done in **GFVxVIPSet()**)
- Host YUV source FIFO (YUV4:2:0 and YUV4:2:2 surfaces only)
- Screen encoding (display source is the primary surface)
- Bit stream encoding. This option re-encodes a JPEG bit stream. It invokes the GFJxDecAPI to decode the bit stream and to send it to the encoder.

  The following options are available for bit stream encoding:

  - Crop image size
  - Scale up

- Scale down (In older GPUs the API may use VI decimation, which restricts sizes to few ratios.)
- Change quantization tables
- Apply rate control to restrict the maximum file size (GoForce 4800 and later)

Digital zoom can be used with any encoding source, depending on the source and destination rectangle sizes. The API performs digital zoom in two passes.

Pass 1. Encode and keep the encoded data in a portion of the output buffer.

Pass 2. Decode the encoded data and route the decoded data to the encoder through a video scalar.

Piecemeal Encoding from the Host

The piecemeal encoding option is intended for encoding large images while using limited host memory resources, although it works for any size image. Only YUV4:2:0 and YUV4:2:2 interleaved formats are currently supported. When the JPEG encoder is set for piecemeal encoding, features like rotation and flip are not supported. Piecemeal host encoding requires that the **`srcRect`** height must be greater than the **`srcSurf`** height. This implies that an application wants to encode a larger image than the system memory surface can hold, so the piecemeal option is used to register a callback function that feeds in pieces of the surface (bands).

GFJxEncEnd()

This function disables the encoder and releases internal resources related to the current capture.

Function Prototype

```
GF_RETTYPE  GFJxEncEnd(
    GF_HANDLE  JXEncHandle );
```

Parameters

`JXEncHandle`	Handle specific to the GFJxEncAPI.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFJxEncInterruptControl()

This function provides component-level interrupt control for the GFJxEncAPI.

Function Prototype

```
GF_RETTYPE  GFJxEncInterruptControl(
    GF_HANDLE                           JxEncHandle,
    GFJX_ENC_INTERRUPT_TYPE             IntType,
    GFJX_ENC_INTERRUPT_OPERATION_TYPE   op,
    void                                *pData );
```

Parameters

`JXEncHandle`	Handle specific to the GFJxEncAPI.
`IntType`	GFJxEncAPI interrupt type as is defined in "GFJX_ENC_INTERRUPT_TYPE" on page 396.
`op`	GFJxEncAPI interrupt operation as is defined in "GFJX_ENC_INTERRUPT_OPERATION_TYPE" on page 396.
`pData`	Pointer to data that is passed in or out by this function. • It is the DMA interrupt control buffer threshold value when `IntType` is `GFJX_ENC_DMA_FIFO_LOW_INTR` or `GFJX_ENC_DMA_FIFO_HIGH_INTR`. • It is the stream buffer threshold value when `IntType` is `GFJX_ENC_STREAM_BUF_INTR` and `op` is `GFJX_ENC_INTERRUPT_SET_BUF_THRESHOLD` or `GFJX_ENC_INTERRUPT_GET_BUF_THRESHOLD`. • It is the interrupt status when `op` is `GFJX_ENC_INTERRUPT_QUERY_STATUS`. • It is the interrupt DMA FIFO threshold value when `op` is `GFJX_ENC_INTERRUPT_SET_DMA_FIFO_THRESHOLD` or `GFJX_ENC_INTERRUPT_GET_DMA_FIFO_THRESHOLD`.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.
`GF_ERROR_NO_SUPPORT`	If the interrupt operation is not supported.

GFJxEncInterruptHandler()

This function is the interrupt handler function for the GFJxEncAPI. It is optional because the interrupt service thread (IST) at the application level can call the GFJxEncAPI to complete the task.

Function Prototype

```
GF_RETTYPE  GFJxEncInterruptHandler(
    GF_HANDLE                JxEncHandle,
    GFJX_ENC_INTERRUPT_TYPE  IntType,
    void                     *pData );
```

Parameters

JXEncHandle	Handle specific to the GFJxEncAPI.
IntType	GFJxEncAPI interrupt type as defined in "GFJX_ENC_INTERRUPT_TYPE" on page 396.
*pData	Pointer to data passed in or out by this function. It is PGFJXENC_FETCH_BUF when IntType is GFJX_ENC_STREAM_BUF_INTR or GFJX_ENC_DONE_INTR.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxEncPRComplete()

This function processes the partially encoded data generated by the hardware encoder to be a fully compliant rotated JPEG bit stream. The application should set partial rotation using **GFJxSetAttribute()** with **JX_ENC_PARTIAL_ROTATE** and then encode the image. The partially encoded data is passed to **GFJxEncPRComplete()** for further processing, and the function returns the fully rotated image in **outBuf**.

Partial rotation is recommended for rotating large images from the camera without needing storage in GPU memory. This function is host-CPU-memory intensive (see "Memory Requirements" on page 390) and MIPS intensive, but it is not performed in real time.

Function Prototype

```
GF_RETTYPE GFJxEncPRComplete()(
    GF_HANDLE     JxEncHandle,
```

Function Prototype (continued)

```
    PGFJXENC_BUF  inBuf,
    NvU32         inBufNum,
    PGFJXENC_BUF  outBuf,
    NvU32         outBufNum,
    PGFJXENC_BUF  dcacBuf,
    NvU32         dcacBufNum );
```

Parameters

`JXEncHandle`	Handle specific to the GFJxEncAPI.
`inBuf`	Array of buffer(s) to hold the partially rotated data that is obtained from `GFJxEncFetchImage()`. See "Buffer Arrays" below.
`inBufNum`	Number of buffers in the `inBuf` array.
`outBuf`	Array of buffer(s) to hold the final image produced by this function.
`outBufNum`	Number of buffers in the `outBuf` array. See "Buffer Arrays" below.
`dcacBuf`	Array of buffer(s) to hold the temporary data (DC/AC information). Each buffer must be a multiple of eight bytes.
`dcacBufNum`	Number of buffers in the `dcacBuf` array.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

Buffer Arrays

In some real-time operating systems, it is hard to allocate contiguous virtual memory because there is no virtual memory manager. That is why **`GFJxEncPRComplete()`** uses arrays of buffers instead of single buffers for **`inBuf`**, **`outBuf`**, and **`dcacBuf`**. If contiguous memory isn't available, it should be possible to allocate multiple buffers and to pass arrays of buffers to **`GFJxEncPRComplete()`**. The function is faster with fewer buffers.

Memory Requirements

`GFJxEncPRComplete()` processes the data in **`inBuf`** and stores it and the header in **`outBuf`**, which contains the final image. Memory requirements are as follows:

- **`inBuf`**. Same size as the compressed image, which is typically a compression ratio of between 1:5 and 1:10.
- **`outBuf`**. Same size as the compressed image.
- **`dcacBuf`**. Its size depends on the number of blocks in the image. Each block in the image occupies eight bytes for temporary information. Here is how to compute the size of **`dcacBuf`** for different image formats (dimensions are width*height):
  - **YUV4:2:0**. Size = 8*6*((width + 15)/16)*((height + 15)/16).

    For a VGA image: size = 8*6*((640 + 15)/16)*((480 + 15)/16) = 57600 bytes.
  - **YUV4:2:2**. size = 8*4*((width + 15)/16)*((height + 7)/8).

    For a VGA image: size = 8*6*((640 + 15)/16)*((480 + 7)/8) = 76800 bytes.
  - If contiguous memory can't be allocated, each buffer must be a multiple of eight bytes.
  - Partial rotation cannot handle image sizes that are not multiples for the minimum coded unit (MCU): 16x16 for YUV4:2:0 and 8x16 for YUV4:2:2).

GFJxEncAPI Data Structures

The GFJxEncAPI data structures include the following:

GFPROPERTY

The structure is described in "GFPROPERTY" on page 10. The GFJxEncAPI **GFPROPERTY** capabilities are defined below.

GFPROPERTY Definitions

```
/*
  Capability Definitions
*/
#define GF_JX_ENC_CAP_QTABLE                  0x00000001
    // Quant table is programmable.
#define GF_JX_ENC_CAP_MAX_BBITS               0x00000002
    // Max Block Bits is programmable.
#define GF_JX_ENC_CAP_INTERRUPT               0x00000004
    // Support interrupt capture mode and polling mode.
    // If this flag is off, only polling mode is supported.
#define GF_JX_ENC_CAP_DIGITAL_ZOOM            0x00000008
    // Support digital zoom.
#define GF_JX_ENC_CAP_MOTION_JPEG             0x00000010
    // Support motion JPEG mode.
#define GF_JX_ENC_CAP_RATE_CONTROL            0x00000020
    // Support rate control.
#define GF_JX_ENC_CAP_422_PLANAR              0x00000040
    // Support 422 planar.
#define GF_JX_ENC_CAP_ROTATION                0x00000080
    // Support rotation in small-size image.
```

GFJXENCSTART

The surface the application wants to encode is **pSrcSurf**. If the application wants to encode an image from a camera or to re-encode a bit stream, this field should be set to NULL. If the application wants to encode an image from the display, this field should point to the primary surface. If an application wants to encode an existing image, it must create a system memory surface to hold the image and set this field to the system memory surface.

GFJXENCSTART Structure

```
typedef struct  _GFJXENCSTART {
    PGFRMSURFACE    pSrcSurf;
    PGFRECT         pSrcRect;
    NvU32           DestWidth;
    NvU32           DestHeight;
```

GFJXENCSTART Structure (continued)

```
    NvU32           uiOptions;
    NvU32           Error;
    void            *pAppPara;
    void            (*pfnGetData)(GF_HANDLE JxHandle,
                     void *pAppPara, PGFRMSURFACE pSurf,
                     PGFRECT pRect, NvU32 options);
} GFJXENCSTART, *PGFJXENCSTART;
```

GFJXENCSTART Fields

`pSrcSurf`	The surface the application wants to encode.
`pSrcRect`	Source image area to be encoded. See "GFRECT" on page 11.
`DestWidth`	Final encoded image width.
`DestHeight`	Final encode image height.
`uiOptions`	See `GFJXENCSTART` definitions below.
`Error`	See `GFJXENCSTART` definitions.
`*pAppPara`	Pointer to application-defined parameter structure.

`(*pfnGetData)`
`(GF_HANDLE JxHandle,`
`void *pAppPtr,`
`PGFRMSURFACE pSurf,`
`PGFRECT pRect,`
`NvU32 options)`
	`pfnGetData`. Callback function to feed the data.
	• `pAppPtr`. Application-defined parameter structure
	• `pSurf`. Pointer to surface to be fed.
	• `pRect`. Rectangle to be fed from the surface.
	• `options`. Not used.

GFJXENCSTART Definitions

```
/*
    uiOptions Definitions
*/
#define GF_CAPT_BITSTREAM                                1
    // Just returns the bit stream if flag is on.
    // If this flag is off, the JFIF header is added.
#define GF_CAPT_PIECE_MEAL                               2
    // If this flag is set, piecemeal encoding from host is
    // enabled. If this feature is used, application must
    // register a callback function and a parameter block
    // to be passed in that function.
```

GFJXENCSTART Definitions (continued)

```
/*
    Error Definitions
*/
#define GF_CAPT_ERROR_DESTSURF_NOT_BIG_ENOUGH         1
    // DestSurf is not big enough.
    // Try to release unnecessary video surfaces.
#define GF_CAPT_ERROR_WRONG_CONFIG                    2
    // Wrong configuration information: srcRect width is
    // greater than srcSurf width.
#define GF_CAPT_ERROR_NO_CALLBACK                     4
    // No Callback function is available in GFJxEncStart()
    // for piecemeal encoding.
```

GFJXENC_BUF

This is a buffer structure that handles the fetched data information between the GFJxEncAPI and the application. It is used when the interrupt callback function is fetching the image.

GFJXENC_BUF Structure

```
typedef struct  _GFJXENC_BUF {
    NvU8   *pBuf;
    NvU32  bufSize;
    NvU32  nBytesCaptured;
} GFJXENC_BUF, *PGFJXENC_BUF;
```

GFJXENC_BUF Fields

`*pBuf`	Pointer to buffer that holds the encoded bit stream.
`bufSize`	Size of the buffer in bytes.
`nBytesCaptured`	Number of bytes captured; application initializes to 0.

GFJXENC_BUF Definitions

```
/*
    uiFetchInfo Definitions
*/
#define JXENC_FETCH_COMPLETE                          0x0
    // GFJxEncAPI has fetched all data in the current
    // encoded frame.
```

GFJXENC_BUF Definitions (continued)

```
#define JXENC_MORE_FETCH                                0x1
    // GFJxEncAPI only fetched part of the data in pBuf.
    // Application must make more calls to fetch the rest.
#define JXENC_NEW_FRAME                                 0x4
    // GFJxEncAPI has encoded a new frame and is ready to
    // fetch it.
```

GFJXENCCALLBACK

This structure is used by **GFJxEncSetAttribute()** on page 377 with attribute **JX_ENC_BITSTREAM**.

GFJXENCCALLBACK Structure

```
typedef struct  _GFJXENCCALLBACK {
    void  *pPara;
    NvU32 (*pCallBack)(void *pPara, NvU8 **ppBuffer,
           NvS32 *BufferLength);
} GFJXENCCALLBACK
```

GFJXENCCALLBACK Fields

*pPara	Parameter that the application passes and wants the GFJxEncAPI to call back with.
(*pCallBack) (void *pPara, NvU8 **ppBuffer, NvS32 *BufferLength);	The GFJxEncAPI uses this callback function to ask the application to pass in the bit stream.

GFJXENC_FETCH_BUF

This data structure is used by **GFJxEncInterruptHandler()** to fetch encoded image data when **IntType** is **GFJX_ENC_STREAM_BUF_INTR** or **GFJX_ENC_DONE_INTR**.

GFJXENC_FETCH_BUF Structure

```
typedef struct _GFJXENC_FETCH_BUF {
    PGFJXENC_BUF  pBuf;
    NvU32         numOfBuf;
    NvU32         status;
}  GFJXENC_FETCH_BUF, *PGFJXENC_FETCH_BUF;
```

GFJX_ENC_INTERRUPT_OPERATION_TYPE

This is the data type for GFJxEncAPI interrupt operations.

GFJX_ENC_INTERRUPT_OPERATION_TYPE Enumeration Type

```
typedef enum {
    GFJX_ENC_INTERRUPT_ENABLE,
    GFJX_ENC_INTERRUPT_DISABLE,
    GFJX_ENC_INTERRUPT_CLEAR,
    GFJX_ENC_INTERRUPT_QUERY_STATUS,
    GFJX_ENC_INTERRUPT_SET_DMA_FIFO_THRESHOLD,
        // Set DMA fifo threshold.
    GFJX_ENC_INTERRUPT_GET_DMA_FIFO_THRESHOLD,
        // Get DMA fifo threshold.
    GFJX_ENC_INTERRUPT_SET_BUF_THRESHOLD,
        // Set stream buffer threshold.
    GFJX_ENC_INTERRUPT_GET_BUF_THRESHOLD
        // Get stream buffer threshold.
}  GFJX_ENC_INTERRUPT_OPERATION_TYPE;
```

GFJX_ENC_INTERRUPT_TYPE

This is the data type for GFJxEncAPI interrupt types.

GFJX_ENC_INTERRUPT_TYPE Enumeration Type

```
typedef enum {
    GFJX_ENC_DMA_FIFO_LOW_INTR =                    0,
        // JPEG read DMA FIFO threshold interrupt/status is
        // asserted if the JPEG read DMA FIFO status is less
        // than the JPEG read DMA FIFO threshold value.
    GFJX_ENC_DMA_FIFO_HIGH_INTR =                 0x1,
        // JPEG read DMA FIFO threshold interrupt/status is
        // asserted if the JPEG read DMA FIFO status is greater
        // than or equal to the JPEG read DMA FIFO threshold
        // value.
    GFJX_ENC_STREAM_BUF_INTR =                    0x2,
        // Stream buffer threshold interrupt.
    GFJX_ENC_MAIN_BUF_HIT_INTR =                  0x4,
        // Main buffer end hit interrupt.
    GFJX_ENC_DONE_INTR =                          0x8,
        // Encoding done interrupt.
```

GFJX_ENC_INTERRUPT_TYPE Enumeration Type (continued)

```
    GFJX_ENC_TRANSFER_DONE_INTR =                0x10,
        // Encoded stream transfer done interrupt.
    GFJX_ENC_MAX_BIT_HIT_INTR =                  0x20,
        // Maximum bit count hit interrupt.
    GFJX_ENC_CIR_BUF_OVERFLOW_INTR =              0x40
        // Circular buffer overflow interrupt.
}  GFJX_ENC_INTERRUPT_TYPE;
```

CHAPTER 11

JPEG Decoder API (GFJxDecAPI)

GFJxDecAPI Reference

The JPEG decoder API (GFJxDecAPI) consists of the functions described under "GFJxDecAPI Functions" on page 399, and the data structures described under "GFJxDecAPI Data Structures" on page 407.

GFJxDecAPI Functions

The GFJxDecAPI functions include the following:

GFJxDecGetProperty()

This function returns a variety of information, including the version of the decoder module. It is a good practice to call this function to query for the GFJxDecAPI version and its capabilities before using the rest of the GFJxDecAPI functions.

Function Prototype

```
GF_RETTYPE  GFJxDecGetProperty(
    GF_HANDLE         JXhandle,
    PGFJXDECPROPERTY  pJXProp );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
pJXProp	Pointer to "GFJXDECPROPERTY Structure" on page 407.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxDecGetStatus()

This function returns the hardware decoder's status.

Function Prototype

```
GF_RETTYPE  GFJxDecGetStatus(
    GF_HANDLE  Jxhandle,
    NvU32      *pStatus );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
pStatus	See "GFJxDecGetStatus() Definitions".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxDecGetStatus() Definitions

```
/*
    pStatus
*/
#define GF_DECODER_IDLE                         0x00000001
    // If hardware decoder is idle.
#define GF_DECODER_BUSY                         0x00000002
    // If hardware decoder is busy.
```

GFJxDecGetImageInfo()

This function retrieves the image's characteristics from the bit stream header.

Function Prototype

```
GF_RETTYPE  GFJxDecGetImageInfo(
    GF_HANDLE          JXhandle,
    PGFJXDECIMAGEINFO  pInfo );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
pInfo	Get image information.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFJxDecSet()

This function sets up a callback function.

Function Prototype

```
GF_RETTYPE  GFJxDecSet(
    GF_HANDLE  JXhandle,
    NvU32      uiFeature,
    void       *pInfo );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.

Parameters (continued)

uiFeature	See "GFJxDecSet() Definition".
*pInfo	Set up decoder option: pInfo is a pointer to "GFJXDECCALLBACK" on page 411.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error, check pError.

GFJxDecSet() Definition

```
#define JX_DEC_SET_READBITSTREAM                1
    // Set up READBITSTREAM callback function.
    // pInfo should point to PGFJXDECCALLBACK.
```

GFJxDecSetAttribute()

This function sets up the GFJxDecAPI options.

Function Prototype

```
GF_RETTYPE  GFJxDecSetAttribute(
    GF_HANDLE  JXhandle,
    NvU32      uiFeature,
    NvU32      *pInfo );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
uiFeature	See "GFJxDecSetAttribute() Definitions".
*pInfo	Pointer to the information buffer. See "GFJxDecSetAttribute() Definitions".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error, check pError.

GFJxDecSetAttribute() Definitions

```
#define JX_DEC_GETIMAGE                          0x00000001
    // Set the get-decoded-image callback function.
    // *pInfo: Points to the structure
    //         "GFJXDECGETIMAGECALLBACK" on page 411.
    //         An application sets this callback function to
    //         retrieve the decoded JPEG image. The decoded
    //         JPEG image size should be set through
    //         GFJXDECSTART.pDestRect. Currently, the GFSDK
    //         outputs the decoded image at 1/4, 1/16, or 1/1
    //         of the original image size.
#define JX_DEC_GETIMAGE_COLORFORMAT          0x00000002
    // Set the get-decoded-image color format.
    // *pInfo: Points to the color format information requested
    //         by the application from the JX_DEC_GETIMAGE
    //         callback function.
    //         The GFSDK only supports these color formats:
    //         GF_SURFACE_YUV420, GF_SURFACE_YUV422, and
    //         GF_SURFACE_PLANNER_YUV422.
    //         Refer to
    //       "GFRMSURFACE Color Format Definitions" on page 38.
#define JX_DEC_PARTIAL_ROTATE                0x00000004
    // Set the Decoder for Partial Rotation.
```

GFJxDecGetAttribute()

This function gets the GFJxDecAPI options.

Function Prototype

```
GF_RETTYPE  GFJxDecGetAttribute(
    GF_HANDLE  JXhandle,
    NvU32      uiFeature,
    NvU32      *pInfo );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
uiFeature	See "GFJxDecGetAttribute() Definitions".
*pInfo	Pointer to the information buffer. See "GFJxDecGetAttribute() Definitions".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error, check pError.

GFJxDecGetAttribute() Definitions

```
#define JX_DEC_GETIMAGE                          0x00000001
    // Get the get-decoded-image callback function.
    // *pInfo: Points to the structure
    //         "GFJXDECGETIMAGECALLBACK" on page 411. See also
    //        "GFJxDecSetAttribute() Definitions" on page 403.
#define JX_DEC_GETIMAGE_COLORFORMAT              0x00000002
    // Get the get-decoded-image color format.
    // *pInfo: Points to the color format information
               requested by the application. See also
   //         "GFJxDecSetAttribute() Definitions" on page 403.
```

GFJxDecStart()

This function should be called before **GFJxDecDecodeImage()**.

Function prototype

```
GF_RETTYPE  GFJxDecStart(
    GF_HANDLE      JXhandle,
    PGFJXDECSTART  pStart );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
pStart	Pointer to "GFJXDECSTART" on page 410

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error, check pError.

GFJxDecDecodeImage()

This function decodes the image and triggers the blit function to display the decoded image.

Function Prototype

```
GF_RETTYPE  GFJxDecDecodeImage(
```

Function Prototype

```
    GF_HANDLE  JXhandle,
    NvU32      uiFlag );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
uiFlag	Set to zero for this version.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If bitstream has error.

GFJxDecEnd()

This function releases internal resources.

Function prototype

```
GF_RETTYPE  GFJxDecEnd(
    GF_HANDLE  JXhandle );
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If bitstream has error.

GFJxDecInterruptControl

This function provides component-level interrupt control for theGFJxDecAPI.

Function prototype

```
GF_RETTYPE GFJxDecInterruptControl(
    GF_HANDLE                          JXhandle,
    GFJX_DEC_INTERRUPT_TYPE            IntType,
    GFJX_DEC_INTERRUPT_OPERATION_TYPE  op,
    void                              *pData);
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
IntType	GFJxDecAPI interrupt type as defined in "GFJX_DEC_INTERRUPT_TYPE" on page 407.
op	GFJxDecAPI inteerupt operation as defined in "GFJX_DEC_INTERRUPT_OPERATION_TYPE" on page 407.
*pData	Pointer to data being passed in or out.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If bitstream has error.

GFJxDecInterruptHandler

This function provides component-level interrupt control for the GFJxDecAPI. It is optional because an interrupt service thread (IST) at the application level can call the GFJxDecAPI to complete the task.

Function prototype

```
GF_RETTYPE GFJxDecInterruptHandler(
    GF_HANDLE                JXhandle,
    GFJX_DEC_INTERRUPT_TYPE  IntType,
    void                    *pData);
```

Parameters

JXhandle	Handle specific to the GFJxDecAPI.
IntType	See "GFJX_DEC_INTERRUPT_TYPE" on page 407.
*pData	Pointer to data passed in or out.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If bitstream has error.

GFJxDecAPI Enumerations

GFJX_DEC_INTERRUPT_OPERATION_TYPE

```
typedef enum
{
    GFJX_DEC_INTERRUPT_ENABLE,
    GFJX_DEC_INTERRUPT_DISABLE,
    GFJX_DEC_INTERRUPT_CLEAR
} GFJX_DEC_INTERRUPT_OPERATION_TYPE; // Interrupt operation
```

GFJX_DEC_INTERRUPT_TYPE

```
typedef enum
{
   GFJX_DEC_DSP_COMMAND_INTR,
}  GFJX_DEC_INTERRUPT_TYPE;
```

GFJxDecAPI Data Structures

The GFJxDecAPI data structures are as follows:

- "GFJXDECPROPERTY" on page 407
- "GFJXDECIMAGEINFO" on page 409
- "GFJXDECSTART" on page 410
- "GFJXDECCALLBACK" on page 411

GFJXDECPROPERTY

GFJXDECPROPERTY is used by "GFJxDecGetProperty()" on page 400.

GFJXDECPROPERTY Structure

```
typedef struct  _GFJXDECPROPERTY {
    NvU32  JXDECVersion;
    NvU32  JXDECCapFlags;
    NvU32  BuildNumber;
    void   *pReserved;
}  GFJXDECPROPERTY;
```

GFJXDECPROPERTY Fields

`JXDECVersion`	Upper 16 bits: major number; lower 16 bits: minor number. For example, `0x00010002` indicates version 1.2
`JXDECCapFlags`	See "GFJXDECPROPERTY Definitions" on page 408.
`BuildNumber`	Build number. For example, `2002093001` means year 2002, month 09, day 30, and 01 indicates the second build.
`*pReserved`	(*Internal use only.*)

GFJXDECPROPERTY Definitions

```
/*
    JXDECCapFlags
*/
#define GF_JXDEC_CAP_SEQUENTIAL_DCT          0x00000001
    // Support sequential DCT-based mode.
#define GF_JXDEC_CAP_PROGRESS_DCT            0x00000002
    // Support progress DCT-based mode.
#define GF_JXDEC_CAP_LOSSLESS                0x00000004
    // Support lossless mode.
#define GF_JXDEC_CAP_HIERARCHICAL            0x00000008
    // Support hierarchical mode.
#define GF_JXDEC_CAP_8_BIT                   0x00000010
    // Support 8 bits.
#define GF_JXDEC_CAP_12_BIT                  0x00000020
    // Support 12 bits.
#define GF_JXDEC_CAP_INTERLEAVE              0x00000040
    // Support interleave mode.
#define GF_JXDEC_CAP_NO_INTERLEAVE           0x00000080
    // Support non-interleave mode.
#define GF_JXDEC_CAP_HUFFMAN                 0x00000100
    // Support Huffman coding.
#define GF_JXDEC_CAP_ARITHMETIC              0x00000200
    // Support arithmetic coding.
#define GF_JXDEC_CAP_INTERCHANGE             0x00000400
    // Support interchange format.
#define GF_JXDEC_CAP_CROPPING                0x00001000
    // Support cropping in MCU base, not pixel base.
#define GF_JXDEC_CAP_YUV444                  0x00002000
    // Support YUV444 format.
#define GF_JXDEC_CAP_YUV422                  0x00004000
    // Support YUV422 format.
```

GFJXDECPROPERTY Definitions (continued)

```
#define GF_JXDEC_CAP_ROTATED_YUV422          0x00008000
    // Support rotated YUV422 format.
#define GF_JXDEC_CAP_YUV420                  0x00010000
    // Support YUV420 format.
#define GF_JXDEC_CAP_GRAY_SCALE              0x00020000
    // Support gray scale format.
#define GF_JXDEC_CAP_EXTEND_RGB              0x00040000
    // Support extended RGB format: 24/32 bpp.
#define GF_JXDEC_CAP_420_OUTPUT              0x00080000
    // Support YUV420 output.
#define GF_JXDEC_CAP_422_OUTPUT              0x00100000
    // Support YUV422 output.
```

GFJXDECIMAGEINFO

Used by **`GFJXDecGetImageInfo()`**.

GFJXDECIMAGEINFO Structure

```
typedef struct  _GFJXDECIMAGEINFO {
    NvU32  IIFlag;
    NvU32  Width;
    NvU32  Height;
    NvU32  ColorFormat;
}  GFJXDECIMAGEINFO;
```

GFJXDECIMAGEINFO Fields

`IIFlag`	See "GFJXDECIMAGEINFO Definition" on page 410.
`Width`	In pixels.
`Height`	In pixels.
`ColorFormat`	Reference to `GFVXSURF.ColorFormat`

GFJXDECIMAGEINFO Definition

```
/*
    IIFlag
*/
#define GF_JXDEC_IFLAG_INVALID_MARKER       0x00000001
#define GF_JXDEC_IFLAG_INVALID_HDR_LEN      0x00000002
#define GF_JXDEC_IFLAG_INVALID_SIG          0x00000004
#define GF_JXDEC_IFLAG_INVALID_DQT          0x00000008
#define GF_JXDEC_IFLAG_INVALID_DHT          0x00000010
#define GF_JXDEC_IFLAG_INVALID_VALUE        0x00000020
#define GF_JXDEC_IFLAG_INVALID_FORMAT       0x00000040
```

GFJXDECSTART

Used by **`GFJxDecStart()`**.

GFJXDECSTART Structure

```
typedef struct  _GFJXDECSTART {
    NvU32          SFlag;
    PGFRMSURFACE   pDestSurf;
    PGFRECT        pCroppingRect;
    PGFRECT        pDestRect;
    NvU32          Error;
}  GFJXDECSTART;
```

GFJXDECSTART Fields

`SFlag`	See "GFJXDECSTART Definitions" on page 410.
`pDestSurf`	Surface to hold the decoded image.
`pCroppingRect`	See `GFJXDECSTART` definitions and "GFRECT" on page 11.
`pDestRect`	Pointer to destination rectangle for decoded image.
`Error`	See `GFJXDECSTART` definitions.

GFJXDECSTART Definitions

```
/*
    SFlag
*/
#define GF_JXDEC_SFLAG_CROPPING              0x00000001
    // Set GFJxDECAPI to do cropping based on pCroppingRect.
/*
    Error
*/
```

GFJXDECSTART Definitions (continued)

```
#define GF_JXDEC_NOT_ENOUGH_VIDEO_MEMORY              1
    // Release more unnecessary surfaces.
#define GF_JXDEC_NOT_SUPPORT_DESTSURFACE_FORMAT       2
    // Can't support this destination surface type.
```

GFJXDECCALLBACK

This structure is used by **`GFJxDecSet()`**.

GFJXDECCALLBACK Structure

```
typedef struct  _GFJXDECCALLBACK {
    void  *pPara;
    NvU32 (*pCallBack)(void *pPara, NvU8 **ppBuffer,
           NvS32 *BufferLength);
}  GFJXDECCALLBACK;
```

GFJXDECCALLBACK Fields

`pPara`	Parameter that application passes and wants the GFJxDecAPI to call back with.
`(*pCallBack) (void *pPara, NvU8 **ppBuffer, NvS32 *BufferLength)`	The GFJxDecAPI uses this callback function to ask the application to pass in the bitstream.

GFJXDECGETIMAGECALLBACK

This structure is used by "GFJxDecGetAttribute()" on page 403 and "GFJxDecSetAttribute()" on page 402.

GFJXDECGETIMAGECALLBACK Structure

```
typedef struct  _GFJXDECGETIMAGECALLBACK {
    void  *pPara;
    NvU32 (*pCallBack)(void * pPara, PGFRMSURFACE pImageSurf,
            PGFRECT pImageRect, NvU32 uiImageFlag);
}  GFJXDECGETIMAGECALLBACK, *PGFJXDECGETIMAGECALLBACK;
```

GFJXDECGETIMAGECALLBACK Fields

`*pPara`	Parameter that application passes and wants the GFJxDecAPI to call back with.

`(*pCallBack)`
`(void * pPara,`
`PGFRMSURFACE pImageSurf,`
`PGFRECT pImageRect,`
`NvU32 uiImageFlag)`
	This function could be called back by the GFJxDecAPI several times. Each time, a slice of the image is ready in `pImageSurf`. The valid image area is specified by `pImageRect`.

`uiImageFlag`	This is the last portion of the decoded image. See "GFJXDECGETIMAGECALLBACK Definition" on page 412

GFJXDECGETIMAGECALLBACK Definition

```
#define JX_DEC_IF_LAST_PORTION_IMAGE          0x00000001
    // Last portion of the decoded image
```

GFJxDecAPI Programming Sequence

Calling **GFRmOpen()** to start GFSDK usage is a prerequisite for using the following procedure.

1. A decoder application should call **GFRmComponentGet()** with **GF_VXAPI** to obtain a Vx Component handle first.
2. Query the properties through **GFVxGetProperty()** to see whether this version of Vx supports JPEG decoding.
3. If JxDec is supported, call **GFRmComponentGet()** with **GF_JXDAPI** to obtain a JxDec Component handle.
4. Query the properties through **GFJxDecGetProperty()** to check whether the desired JxDec features can be supported.
5. Call **GFJxDecGetImageInfo()** to get the JPEG image information, such as width and height.
6. Call **GFRmSurfaceAlloc()** to allocate one surface to hold the decoded image.
7. Enable the Auto Blt feature when calling **GFVxBlt()** function to set up color space conversion, stretching, and shrinking.
8. The application needs to call **GFJxDecSet()** to set up various decoding options.
9. **GFJxDecStart()** is the next function to call. It starts the decoding process before the bitstream is fed to the decoder.
10. Call **GFJxDecDecodeImage()** to do the real decoding. The application may call this function several times while the bitstream is being fed.
11. After the whole bitstream has been fed, call **GFJxDecEnd()** to end the decoding process.
12. Before exiting the application, the application should call **GFRmSurfaceFree()** to free all allocated surfaces.
13. Call **GFRmComponentRelease()** with the JxDec handle to release all of the resources of the decoder module.
14. Lastly, call **GFRmComponentRelease()** with the Vx handle to release all of the resources of the video engine.

CHAPTER

12

MPEG-4 Encoder API (GFMxEncAPI)

This chapter discusses the MPEG-4 encoder API (GFMxEncAPI), which is described in the next section "GFMxEncAPI Reference" and in "GFMxEncAPI Programming Assistance" on page 436. It also discusses the MPEG-4 encoder interrupt API, detailed in "MPEG-4 Encoder Interrupt API " on page 443.

GFMxEncAPI Reference

The MPEG-4 encoder API is for GoForce 4000 (and newer) media processors and consists of the functions described under "GFMxEncAPI Functions" on page 415, and the data structures described under "GFMxEncAPI Data Structures" on page 426.

GFMxEncAPI Functions

The GFMxEncAPI functions include the following:

- "GFMxEncGetProperty()" on page 416
- "GFMxEncGetStatus()" on page 416
- "GFMxEncSetVOL()" on page 417

GFMxEncGetProperty()

This function returns a variety of information, including the version of the GFMxAPI encoder module. It is a good practice to call this function to query for the GFMxEncAPI version and its capabilities before using the rest of the GFMxEncAPI functions.

Function Prototype

```
GF_RETTYPE  GFMxEncGetProperty(
    GF_HANDLE    MXhandle,
    PGFPROPERTY  pMXProp );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
pMXProp	Pointer to GFPROPERTY.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncGetStatus()

This function returns the hardware encoder's status.

Function Prototype

```
GF_RETTYPE  GFMxEncGetStatus(
```

Function Prototype (continued)

```
    GF_HANDLE  MXhandle,
    NvU32     *pStatus );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
*pStatus	See "GFMxEncGetStatus Definitions".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncGetStatus Definitions

```
/*
    pStatus
*/
#define GF_ENCODER_BUSY                         0x00000001
    // If hardware encoder is busy.
#define GF_ENCODER_VOP_READY                    0x00000002
    // At least one VOP is ready for fetching.
```

GFMxEncSetVOL()

This function sets the characteristics of a new video object layer (VOL) and tells the GFMxEncAPI the VOL information. The GFMxEncAPI does not generate a VOL bit stream. The application needs to prepare a VOL by itself. Please refer to "GFMxEncAPI Programming Assistance" on page 436.

Function Prototype

```
GF_RETTYPE  GFMxEncSetVOL(
    GF_HANDLE    MXhandle,
    PGFMXENCVOL  pVOL );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
pVOL	Set new VOL information.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncSetVOP()

This function sets the characteristics of a new video object plane (VOP). This function should be called after **GFMxEncSetVOL()** and before **GFMxEncStart()**. If the application decides to change any VOP information during the encoding time (after **GFMxEncStart()**), the application can call this function to pass the new VOP information to the GFMxEncAPI. Otherwise, the application should not call this function, allowing the GFMxEncAPI to reduce overhead.

Function Prototype

```
GF_RETTYPE  GFMxEncSetVOP(
    GF_HANDLE    MXhandle,
    PGFMXENCVOP  pVOP );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
pVOP	Set new VOP information.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncRateControlConfig()

This function initializes the GFMxEncAPI rate control block. This function is optional for application because the GFMxEncAPI automatically initializes the rate control block.

Function Prototype

```
GF_RETTYPE  GFMxEncRateControlConfig(
    GF_HANDLE    MXhandle,
    PGFMXENCRCC  pRCC );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
pRCC	Set rate control information.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncFeedImage()

The application can call this function if the application has the source image. This function should not be called if the image comes directly from the video input port (VIP), for example, from a camera.

Function Prototype

```
GF_RETTYPE  GFMxEncFeedImage(
    GF_HANDLE           MXhandle,
    PGFMXENCFEEDIMAGE   pFeedImage );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
pFeedImage	New image information.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncFetchImage()

An application should call this function to fetch the encoded VOP bit stream or encoded raw data. If the source image is from the host, the application should call this function after **GFMxEncFeedImage()**. If the source image is from a video camera connected to the VIP and the application is using a polling scheme, the application should call this function at least at the camera's frame rate.

Function Prototype

```
GF_RETTYPE  GFMxEncFetchImage(
```

Function Prototype (continued)

```
    GF_HANDLE          MXhandle,
    PGFMXENCFETCHVOP   pFetchVOP );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
pFetchVOP	Structure to hold encoded VOP information.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncStart()

This function starts the GFMxEncAPI module for encoding the bit stream. The GFMxEncAPI assumes the first VOP it outputs is the first VOP in the current VOL.

Function Prototype

```
GF_RETTYPE  GFMxEncStart(
    GF_HANDLE  MXhandle );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncPause()

This function pauses the encoding process until **GFMxEncStart()** is called to restart encoding, or **GFMxEncStop()** is called to totally stop encoding.

Function Prototype

```
GF_RETTYPE  GFMxEncPause(
    GF_HANDLE  MXhandle );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncStop()

This function stops the encoding process.

Function Prototype

```
GF_RETTYPE  GFMxEncStop(
    GF_HANDLE  MXhandle );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncSetupInterrupt() [Obsolete]

Note: This function is obsolete. Please use the general interrupt framework (see "Interrupt Architecture API (GFINTxAPI)" on page 109) and connect to the appropriate interrupt. Most interrupt processing code resides on the application side.

An application can use this function to set up the interrupt callback function. This function must be called before **GFMxEncStart()**.

If this function returns **GF_SUCCESS**, whenever a VOP is ready the GFMxEncAPI calls the interrupt callback function. Inside of the callback function, the application can call **GFMxEncSetVOP()** if it is needed, and call **GFMxEncFetchImage()** to fetch the encoded VOP.

If **GF_ERROR** is returned, the application should use a polling scheme.

Function Prototype

```
GF_RETTYPE  GFMxEncSetupInterrupt(
    GF_HANDLE  MXhandle,
    void       (*Inter)(void*)
    void       *IPara  );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
(*Inter)(void*)	Pointer to encoder application's interrupt callback function.
*IPara	Pointer to parameter of encoder application's interrupt callback function.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncSetAttribute()

This function sets a feature of the MPEG encoder hardware.

Function Prototype

```
GF_RETTYPE GFMxEncSetAttribute(
    GF_HANDLE  MXhandle,
    NvU32      uiFeature,
    NvU32      *pInfo  );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
uiFeature	See "GFMxEncSetAttribute Definitions".
*pInfo	Pointer to the information buffer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncSetAttribute Definitions

```
/*
    Attribute Definition
*/
#define MXENC_ATTR_ROTATION                                1
    // GFMxEncAPI rotation option.
/*
    *pInfo Degree of Rotation
*/
#define MXENC_ATTR_ROTATE_0                              0x0
    // Rotate to 0 degrees.
#define MXENC_ATTR_ROTATE_90                             0x1
    // Rotate to 90 degrees.
#define MXENC_ATTR_ROTATE_180                            0x2
    // Rotate to 180 degrees.
#define MXENC_ATTR_ROTATE_270                            0x3
    // Rotate to 180 degrees.
/*
    Horizontal flip, vertical flip, and one of the rotations
    can be enabled at the same time.
*/
#define MXENC_ATTR_H_FLIP                               0x10
    // Horizontal flip.
#define MXENC_ATTR_V_FLIP                               0x20
    // Vertical flip.
#define MXENC_ATTR_KEEP_PREV_ENCODING_INTERNAL_RESOURCE   2
    // *pInfo 0:
    //  Release any internal resources (buffers) of the
    //  previous encoding process. Allocate new resources
    //  for the current encoding process.
    // *pInfo 1:
    //  Application wants the GFMxEncAPI to re-use any
    //  internal resources of the previous encoding process.
    //  If the attribute is on and the new encoding process
    //  needs more resources, the GFMxEncAPI releases the
    //  previous internal resources and re-allocates new
    //  ones. If the new encoding process needs equal or
    //  fewer resources, the GFMxEncAPI uses previous
    //  internal resource. If there was no previous encoding
    //  process, the GFMxEncAPI allocates new resources.
```

GFMxEncSetAttribute Definitions (continued)

```
#define MXENC_PREVIEW_ROTATE_BUFFER                          3
    // GoForce 4800 supports rotation for video encoding.
    // In the rotated mode, a full frame of video has to be
    // captured before the encoder can start encoding.
    // The YUV4:2:0 data in the rotated buffer can be used
    // for previewing also.
    // This option can be used to enable the preview from
    // these buffers. In rotated mode, the encoder has to
    // be on, even for preview.
    // VI auto trigger does not work if the VI is not
    // writing to a YUV4:2:2 memory buffer.
    // GFMxEncPause(): In the rotated mode, an application
    // should call this function right after video input
    // Vsync to avoid tearing in the recording.
    // *pInfo = 1: enables preview from the rotated buffers.
    // *pInfo = 0: disables this mode (normal mode).
    // This option can only be used when MXENC_ATTR_ROTATION
    // is set and/or the GFMxEncSetVOL() option
    // GF_MXENC_VOL_PREPARE_ROTATION is set.
#define MXENC_GET_PREVIEW_BUFFER                             4
    // Option is only supported for GFMxEncGetAttribute().
    // Returns an error for GFMxEncSetAttribute().
```

GFMxEncGetAttribute()

This function gets a feature of the MPEG encoder hardware.

Function Prototype

```
GF_RETTYPE GFMxEncGetAttribute(
    GF_HANDLE  MXhandle,
    NvU32      uiFeature,
    NvU32      *pInfo  );
```

Parameters

MXhandle	Handle specific to the GFMxEncAPI.
uiFeature	See "GFMxEncGetAttribute Definitions".
*pInfo	Pointer to the information buffer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncGetAttribute Definitions

```
/*
    Attribute Definition
*/
#define MXENC_ATTR_ROTATION                              1
    // GFMxEncAPI rotation option.
/*
    *pInfo Degree of Rotation
*/
#define MXENC_ATTR_ROTATE_0                            0x0
    // Rotate to 0 degrees.
#define MXENC_ATTR_ROTATE_90                           0x1
    // Rotate to 90 degrees.
#define MXENC_ATTR_ROTATE_180                          0x2
    // Rotate to 180 degrees.
#define MXENC_ATTR_ROTATE_270                          0x3
    // Rotate to 180 degrees.
#define MXENC_ATTR_H_FLIP                             0x10
    // Horizontal flip.
#define MXENC_ATTR_V_FLIP                             0x20
    // Vertical flip.
#define MXENC_ATTR_KEEP_PREV_ENCODING_INTERNAL_RESOURCE   2
    // *pInfo 0: Attribute is off.
    // *pInfo 1: Attribute is on.
#define MXENC_PREVIEW_ROTATE_BUFFER                       3
    // Gets the status of this attribute.
    // Returns *pInfo = 1 if preview from the rotated
    // buffers is enabled.
    // Returns *pInfo = 0 if preview from the rotated
    // buffers is disabled (normal mode).
#define MXENC_GET_PREVIEW_BUFFER                          4
    // Rotated buffers are allocated in the GPU memory by the
    // API. Previewing is normally done by an application
    // using GFVxBlt() to convert an off-screen YUV surface
    // to an on-screen RGB rectangular area.
```

GFMxEncAPI Data Structures

The GFMxEncAPI data structures include the following:

GFPROPERTY

This structure is described in "GFPROPERTY" on page 10. The GFMxEncAPI **`GFPROPERTY`** capabilities are defined below.

GFPROPERTY Definitions

```
/*
    GFPROPERTY Capability Definitions
*/
#define GF_MXENC_CAP_SIMPLE                  0x00000001
    // Support simple profile.
#define GF_MXENC_CAP_SCALE                   0x00000002
    // Support simple scale profile.
#define GF_MXENC_CAP_CORE                    0x00000004
    // Support core profile.
#define GF_MXENC_CAP_MAIN                    0x00000008
    // Support MAIN profile.
#define GF_MXENC_CAP_N_BIT                   0x00000010
    // Support N-bit profile.
#define GF_MXENC_CAP_LEVEL1                  0x00000100
    // Support LEVEL 1.
#define GF_MXENC_CAP_LEVEL2                  0x00000200
    // Support LEVEL 2.
#define GF_MXENC_CAP_LEVEL3                  0x00000400
    // Support LEVEL 3.
#define GF_MXENC_CAP_LEVEL4                  0x00000800
    // Support LEVEL 4.
#define GF_MXENC_CAP_DATA_PARTITIONING       0x00001000
    // Support data partitioning mode.
#define GF_MXENC_CAP_RVLC                    0x00002000
    // Support RVLC mode.
```

GFPROPERTY Definitions (continued)

```
#define GF_MXENC_CAP_RATE_CONTROL            0x00004000
    // Support rate control.
#define GF_MXENC_CAP_IMRF                   0x000008000
    // Support intra macroblock refreshing.
#define GF_MXENC_CAP_ROTATION                0x00010000
    // Support rotation encoding. The GFMxEncAPI rotates the
    // incoming image and then encodes it.
```

GFMXENCVOL

GFMXENCVOL describes a visual object layer (VOL).

GFMXENCVOL Structure

```
typedef struct  _GFMXENCVOL {
    NvU32         uiVOLInfo;
    PGFRMSURFACE  pSrcSurf;
    PGFRECT       pSrcRect;
    NvU16         VOPWidth;
    NvU16         VOPHeight;
    float         fFrameRate;
    NvU16         uiMaxBitRate;
    NvU32         uiBufferSize;
    NvU32         uiTimeIncResolution;
    NvU32         uiTimeStampSource;
    PGFVXSURF     pEncSurf;
} GFMXENCVOL;
```

GFMXENCVOL Fields

uiVOLInfo	VOL characteristics.
pSrcSurf	Surface to be encoded. If the source image is from a camera, set this field to NULL.
pSrcRect	Area of the source image that needs to be encoded.
VOPWidth	Width in pixels.
VOPHeight	Height in pixels.
fFrameRate	Frame rate per second.
uiMaxBitRate	Kilobits/second.

GFMXENCVOL Fields (continued)

`uiBufferSize`	Size in DWORDs. This buffer is created and managed by the application to temporarily hold the encoded bit stream before transferring the bit stream out.
`uiTimeIncResolution`	For nonshort video header only.

GFMXENCVOL Definitions

```
/*
    uiVOLInfo
*/
#define GF_MXENC_VOL_SHORT_VIDEO_HEADER       0x00000001
    // GFMxEncAPI should generate a short video header
    // format VOP. If this flag is off, GFMxEncAPI generates
    // a nonshort video header format VOP.
#define GF_MXENC_VOL_BYPASS_VLC               0x00000002
    // GFMxEncAPI should output raw data, bypassing VLC. If
    // this flag is off, API outputs a VLC bit stream.
#define GF_MXENC_VOL_RESYNC_MARK_ENABLE       0x00000004
    // GFMxEncAPI should put resync mark in the bit stream.
#define GF_MXENC_VOL_DATA_PARTITION_ENABLE    0x00000008
    // Enable data partition.
#define GF_MXENC_VOL_REVERSIBLE_VLC_ENABLE    0x00000010
    // Enable RVLC.
#define GF_MXENC_VOL_RATE_CONTROL_ENABLE      0x00000020
    // Enable rate control.
#define GF_MXENC_VOL_STUFFING_ENABLE        (0x00000040|
        GF_MXENC_VOL_RATE_CONTROL_ENABLE)
    // Stuff more bits to prevent buffer underflow.
#define GF_MXENC_VOL_AUTO_IMRF_ENABLE         0x00000080
    // Enable intra macroblock refreshing. This mode can run
    // automatically without application interaction.
    // Application can change the macroblock's counter
    // range by calling GFMxEncSetVOP().
#define GF_MXENC_VOL_MC_IMRF_ENABLE           0x00000100
    // Enable intra macroblock refreshing with customized
    // counter matrix from application. When this flag
    // is on, application must call GFMxEncSetVOP() to
    // set up intra macroblock refresh counter matrix.
```

GFMXENCVOP

This structure holds video object plane (VOP) information.

GFMXENCVOP Structure

```
typedef struct  _GFMXENCVOP {
    NvU32  uiVOPInfo;
    NvU16  uiPacketSize;
    NvU16  uiIntra_DC_VLC_Thr;
    NvU16  uiHECNum;
    NvU16  uiExpectedISize;
    NvU16  uiExpectedPSize;
    NvU16  uiInitialIQP;
    NvU16  uiInitialPQP;
    NvU16  uiMaxIQP;
    NvU16  uiMinIQP;
    NvU16  uiMaxPQP;
    NvU16  uiMinPQP;
    NvU16  uiNumofP;
    NvU16  uiRNumerator;
    NvU16  uiRDenominator;
    NvU8   uiIMRFIMinValue;
    NvU8   uiIMRFIMaxValue;
    NvU8   uiIMRFPMinValue;
    NvU8   uiIMRFPMaxValue;
    NvU8   uiIMRFDefaultCounter;
    NvU8   *pMIRFMatrix;
} GFMXENCVOP;
```

GFMXENCVOP Fields

uiVOPInfo	VOP characteristics. See GFMXENCVOP definitions.
uiPacketSize	If GF_MXENC_VOP_PACKET_BASED_ON_MB_NUM flag is on, this field gives number of macroblocks in a packet. Otherwise, the number of bytes in a packet.
uiIntra_DC_VLC_Thr	Intra DC VLC threshold.
uiHECNum	Frequency to insert HEC marker. 0 is no HEC marker. 1–7 is insert HEC after 1–7 resync mark(s).
uiExpectedISize	If the next encoded VOP is I, this field tells the GFMxEncAPI the size in DWORDs of the bit stream the application expects.

GFMXENCVOP Fields (continued)

uiExpectedPSize	If next encoded VOP is P, field tells the GFMxEncAPI the size in DWORDs of bit stream that application expects.
uiInitialIQP	Initial quantization parameter (QP) value for I VOP.
uiInitialPQP	Initial QP value for P VOP.
uiMaxIQP	Maximum QP value for I VOP.
uiMinIQP	Minimum QP value for I VOP.
uiMaxPQP	Maximum QP value for P VOP.
uiMinPQP	Minimum QP value for P VOP.
uiNumofP	Number P VOPs between two I VOPs. If GF_MXENC_VOP_IMRF_CONTROL_IP_RATIO is on, this field has no effect.
uiRNumerator	Number of frames to keep in one group of frames. Maximum value is 24. Field must be filled if flag GF_MXENC_VOP_REDUCE_FRAME_RATE is on.
uiRDenominator	Group of frames. This field must be filled if flag GF_MXENC_VOP_REDUCE_FRAME_RATE is on. Maximum number is 24
uiIMRFIMinValue uiIMRFIMaxValue	Those values take effect only when the GF_MXENC_VOP_LOAD_IMRF_COUNTER_RANGE flag is set. Whenever an I VOP is encoded, the GFMxEncAPI generates a random number in this range for each macroblock.
uiIMRFPMinValue uiIMRFPMaxValue	Those values take effect only when the GF_MXENC_VOP_LOAD_IMRF_COUNTER_RANGE flag is set. Whenever a macroblock counter is reduced to 0, an Intra macroblock is generated by the GFMxEncAPI. The GFMxEncAPI generates a random number in this range for this macroblock.
uiIMRFDefaultCounter	This value is used as the new counter when the refresh counter in the matrix is decreased to 0.
*pMIRFMatrix	Pointer to an array that holds the counter for each macroblock. When this counter decreased to 0, one Intra macroblock is inserted.

GFMXENCVOP Definitions

```
/*
    uiVOPInfo
*/
#define GF_MXENC_VOP_4MV_ENABLE              0x00000001
    // Enable 4 motion vectors. GFMxEncAPI picks 1 MV or
    // 4 MV, depending on which mode is more efficient.
#define GF_MXENC_VOP_HALF_PEL_ENABLE         0x00000002
    // Enable half-pel motion estimation search.
#define GF_MXENC_VOP_PACKET_BASED_ON_MB_NUM  0x00000004
    // New packet is based on the number of macroblocks, not
    // the length of the bit stream.
#define GF_MXENC_VOP_AC_PRE_ALWAYS_ON        0x00000008
    / /AC prediction is always on.
#define GF_MXENC_VOP_AC_PRE_DYNAMICALLY_ON   0x00000010
    // AC prediction is dynamically on.
#define GF_MXENC_VOP_CHANGE_EXPECTED_SIZE    0x00000020
    // Change the expected size for I and P.
    // Must fill uiExpectedISize and uiExpectedPSize.
#define GF_MXENC_VOP_CHANGE_INITQP           0x00000040
    // Change the initial QP for I and P.
    // Must fill uiInitialIQP and uiInitialPQP.
    // NOTE: Not effective when rate control is enabled (by
    // GF_MXENC_VOL_RATE_CONTROL_ENABLE).
#define GF_MXENC_VOP_CHANGE_MIN_MAX_QP       0x00000080
    // Change minimum and maximum QP for I and P. uiMaxIQP,
    // uiMinIQP, uiMaxPQP, and uiMinPQP must be filled.
#define GF_MXENC_VOP_REDUCE_FRAME_RATE       0x00000100
    // Reduce the frame rate by skipping certain frames.
    // Must set uiRNumerator and uiRDenominator.
#define GF_MXENC_VOP_ENCODE_I_ASAP           0x00000200
    // Encode I VOP as soon as possible.
#define GF_MXENC_VOP_IMRF_CONTROL_IP_RATIO   0x00000400
    // Number of P VOPs between two I VOPs is controlled by
    // intra macroblock refreshing. If this flag is off,
    // it is controlled by uiNumofP.
#define GF_MXENC_VOP_LOAD_IMRF_COUNTER_RANGE 0x00000800
    // Application changes the counter range by setting this
    // flag and giving the range in uiIMRFIMinValue,
    // uiIMRFIMaxValue, uiIMRFPMinValue, and
    // uiIMRFPMaxValue. Applications can turn on this flag
    // only when GF_MXENC_VOL_AUTO_IMRF_ENABLE is on.
```

GFMXENCVOP Definitions (continued)

```
#define GF_MXENC_VOP_LOAD_IMRF_MATRIX         0x00001000
    // Application must fill in uiIMRFDefaultCounter and
    // pIMRFMatrix if this flag is on. Application can turn
    // on flag only if GF_MXENC_VOL_MC_IMRF_ENABLE is on.
#define GF_MXENC_VOP_TIMESTAMP_CPU              0x00010000
    // Application must fill in uiIMRFDefaultCounter and
    // pIMRFMatrix if this flag is on. Application can turn
    // on flag only if GF_MXENC_VOL_MC_IMRF_ENABLE is on.
#define GF_MXENC_VOP_TIMESTAMP_VI               0x00020000
    // Application must fill in uiIMRFDefaultCounter and
    // pIMRFMatrix if this flag is on. Application can turn
    // on flag only if GF_MXENC_VOL_MC_IMRF_ENABLE is on.
```

GFMXENCRCC

This structure is used by "GFMxEncRateControlConfig()" on page 418.

GFMXENCRCC Structure

```
typedef struct  _GFMXENCRCC {
    NvU16  uiMinIQP;
    NvU16  uiMaxIQP;
    NvU16  uiMinISize;
    NvU16  uiMaxISize;
    NvU16  uiSuggestedISize;
    NvU16  uiMinPQP;
    NvU16  uiMaxPQP;
    NvU16  uiMinPSize;
    NvU16  uiMaxPSize;
    NvU16  uiSuggestedPSize;
    NvU16  uiUnderFlowThr;
    NvU16  uiOverFlowThr;
    NvU16  uiSkipVOPThr;
} GFMXENCRCC;
```

GFMXENCRCC Fields

uiMinIQP	Minimum QP value for I.
uiMaxIQP	Maximum QP value for I.
uiMinISize	Minimum I VOP size in DWORDs.
uiMaxISize	Maximum I VOP size in DWORDs.

GFMXENCRCC Fields (continued)

uiSuggestedISize	Suggested I VOP size in DWORDs.
uiMinPQP	Minimum QP value for P.
uiMaxPQP	Maximum QP value for P.
uiMinPSize	Minimum P VOP size in DWORDs.
uiMaxPSize	Maximum P VOP size in DWORDs.
uiSuggestedPSize	Suggested P VOP size in DWORDs.
uiUnderFlowThr	Underflow threshold in DWORDs.
uiOverFlowThr	Overflow threshold in DWORDs.
uiSkipVOPThr	Skip VOP if the buffer is over this threshold.

GFMXENCFEEDIMAGE

This structure is used by "GFMxEncFeedImage()" on page 419.

GFMXENCFEEDIMAGE Structure

```
typedef struct  _GFMXENCFEEDIMAGE {
    NvU32         uiTime;
    PGFRMSURFACE  pImgSurf;
} GFMXENCFEEDIMAGE;
```

GFMXENCFEEDIMAGE Fields

uiTime	In milliseconds. First VOP may start from 0.
pImgSurf	Image to be encoded. Application should set the GF_SURFACE_SYSTEM_MEMORY flag to on when it calls GFRmSurfaceAlloc(). GFMxEncAPI can support the GF_SURFACE_YUV420, GF_SURFACE_YUYV, GF_SURFACE_YVYU, GF_SURFACE_UYVY, and GF_SURFACE_VYUY formats. Refer to "Resource Manager Services (GFRm)" on page 17 for details.

GFMXENCFETCHVOP

This structure is used by "GFMxEncFetchImage()" on page 419.

GFMXENCFETCHVOP Structure

```
typedef struct  _GFMXENCFETCHVOP {
    void   *pBuf;
    NvU32  uiSizeofBuf;
    NvU16  *pPacketLengthBuf;
```

GFMXENCFETCHVOP Structure (continued)

```
    NvU32  uiSizeofPLB;
    NvU32  uiUTContentSize;
    NvU32  uiTimeOut;
    NvU32  uiVOPInfo;
    NvU32  uiAverageQP;
    NvU32  uiTime;
    NvU32  uiFetchedSize;
    NvU32  uiPLBSize;
} GFMXENCFETCHVOP, *PGFMXENCFETCHVOP;
```

GFMXENCFETCHVOP Fields

Filled in by Application	
*pBuf	If GF_MXENC_VOL_BYPASS_VLC is set, the data in this buffer is a GFMxEncAPI compressed format bit stream. Otherwise the data is an MPEG-4 compliant bit stream. See "GFMxEncAPI Programming Assistance" on page 436 for this format.
uiSizeofBuf	Number of bytes.
*pPacketLengthBuf	Packet length in bytes. If encoder application wants to know each packet's size, this field should be set. If field is NULL, the GFMxEncAPI does not output any packet size information.
uiSizeofPLB	Number of NvU16.
uiUTContentSize	The size in DWORDs of the bit stream that has not been transferred out of the application's bit stream buffer. Valid only when the application enables rate control. For a source image from the CPU, the application must set this field based on the ideal transfer rate because the application may flush the whole buffer before feeding and fetching another VOP. If more than half of uiBufferSize in GFMXENCVOL has not been transferred out, the GFMXEncAPI starts reducing the bit rate. If less than half, the GFMXEncAPI starts increasing the bit rate.
uiTimeOut	In milliseconds. If 0, return immediately if there is no VOP ready. If -1, wait until VOP is ready.
Filled in by the GFMxEncAPI	
uiVOPInfo	See "GFMXENCFETCHVOP Definitions".

GFMXENCFETCHVOP Fields (continued)

uiAverageQP	Average QP for current frame.
uiTime	In milliseconds, the encoded VOP's time stamp. First VOP may start from 0.
uiFetchedSize	Number of available bytes.
uiPLBSize	Number of available NvU16 in pPacketLengthBuf.

GFMXENCFETCHVOP Definitions

```
/*
    uiVOPInfo
*/
#define MXENC_VOP_P_VOP                                0x1
    // The encoded frame is P VOP. If this flag is off,
    // this VOP is I VOP.
#define MXENC_VOP_MORE_FETCH                           0x2
    // GFMxEncAPI only fed part of VOP data into pBuf and
    // the application must call to fetch the rest.
#define MXENC_VOP_PORTION_PACKET                   (0x4 |
        MXENC_VOP_MORE_FETCH)
    // The last packet in pBuf only contains part of the
    // packet data. The last entry in pPacketLengthBuf is
    // the size of this partial packet data size.
    // The application must call to fetch the rest of data
#define MXENC_VOP_MORE_VOP                             0x8
    // More VOPs are ready in the GFMxEncAPI internal
    // buffer. Application should fetch as soon as possible.
#define MXENC_VOP_BEGIN_VOP                           0x10
    // Fetched bit stream is first portion of current VOP.
#define MXENC_VOP_END_VOP                             0x20
    // Fetched bit stream is last portion of current VOP.
```

GFMxEncAPI Programming Assistance

The following sections are provided to show effective use of the GFMxEncAPI.

GFMxEncAPI Programming Sequence

1. The encoder application should call **`GFRmComponentGet()`** with **`GF_VXAPI`** to obtain a GFVxAPI **`VXhandle`** first.

Note: It's assumed that **`GFRmOpen()`** was called earlier to start GFSDK usage.

---

2. Query the properties with **`GFVxGetProperty()`** to see whether this GFVxAPI version supports MPEG encoding.
3. If it supports MPEG decoding, call **`GFRmComponentGet()`** again with **`GF_MXEAPI`** to obtain a GFMxEncAPI **`MXhandle`**.
4. Call **`GFMxEncGetProperty()`** to query properties. Check whether this version can support the desired MPEG profile and level.
5. If supported, call **`GFRmSurfaceAlloc()`** to allocate one surface for encoding purposes.
6. The encoder application should call **`GFMxEncSetVOL()`** to specify VOL level characteristics.
7. Calling **`GFMxEncRateControlConfig()`** is optional. If the application enables rate control by calling **`GFMxEncSetVOL()`**, the GFMxEncAPI automatically configures the rate control. Otherwise, the application may want to handle rate control by itself.
8. The encoder application should call **`GFMxEncSetVOP()`** at least once to set up VOP level characteristic. If there is no change to the options during the subsequent encoding process, the application does not need to call this function again.
9. Call **`GFMxEncStart()`** to start the encoding process.

10. If the current source image is from the CPU, the application should call **`GFMxEncFeedImage()`** to feed the image to the GFMxEncAPI. If the current source image is coming from the VIP (for example. camera input), the application should skip the **`GFMxEncFeedImage()`** call.
11. If the application uses an interrupt scheme, it must implement a callback function. Inside of the callback function, the application should call **`GFMxEncFetchImage()`** to retrieve the encoded VOP bit stream or the NVIDIA specific compressed bit stream.

    If the application requires a polling scheme, it must call **`GFMxEncFetchImage()`** to retrieve the encoded VOP bit stream or the NVIDIA-specific compressed bit stream. The encoder application should keep encoding, VOP by VOP.
12. Before exiting, the encoder application should call **`GFMxEncStop()`** to stop encoding.
13. Call **`GFRmComponentRelease()`** with the **`MXhandle`** handle to release the resources of the encoder module.
14. The application should call **`GFRmSurfaceFree()`** to free the allocated surfaces.
15. Lastly, call **`GFRmComponentRelease()`** with the **`VXhandle`** handle to release all of the resources of the video engine.

Inserting User Data in VOP Bit Stream

This version of the GFMxEncAPI does not support inserting user date in the VOP bit stream.

NVIDIA Compressed Bit Stream

If the application turns off VLC (bypasses VLC), the GFMxEncAPI output is the NVIDIA compressed bit stream. The format is based on a macroblock-by-macroblock process. Each macroblock has three portions of information. The first portion is macroblock information. The second portion is motion vectors if indicated by the macroblock information. The third portion is for coefficients if indicated by the macroblock information.

Compressed Bit Stream Pseudocode

```
#define GFMX_ENC_CF_INTER                                   0x1
#define GFMX_ENC_CF_CR_CODED                                0x2
#define GFMX_ENC_CF_CB_CODED                                0x4
```

Compressed Bit Stream Pseudocode (continued)

```
#define GFMX_ENC_CF_Y3_CODED                              0x8
#define GFMX_ENC_CF_Y2_CODED                             0x10
#define GFMX_ENC_CF_Y1_CODED                             0x20
#define GFMX_ENC_CF_Y0_CODED                             0x40
#define GFMX_ENC_CF_MV_CODED                             0x80
#define GFMX_ENC_CF_4MV_CODED                           0x100
#define GFMX_ENC_CF_AC_PRED_ENABLE                      0x200
#define GFMX_ENC_CF_NEW_PACKET                          0x400
#define GFMX_ENC_CF_QP_SHIFT                               11
#define GFMX_ENC_CF_QP_MASK                              0x1f
#define GFMX_ENC_CF_MV_MASK                              0x7f
#define GFMX_ENC_CF_COEF_TWO_BYTES                       0x80
#define GFMX_ENC_CF_COEF_LAST                            0x40
#define GFMX_ENC_CF_COEF_RUN_MASK                        0x1f

GFMXENCFETCHVOP *lpRawInfo;

GFMxEncCompressedBitStream()
{
    NvU8 * pOutput = (NvU8 *)(lpRawInfo->pBuf);
    If (*(lpRawInfo->uiVOPInfo) & MXENC_P_VOP)
    {
        vop_fcode = 1;
        vop_rounding_type = 0;
    }
    for (i = 0; i < totalMBsInVOP; i++)
    {
        MacroBlock()
    }
}

MacroBlock()
{
    NvU16 MBInfo = (NvU16)((NvU16*)pOutput);
        // Read first 16 bits output.
    pOutput+=2;
    NvU8 QPStep = (MBInfo >> GFMX_ENC_CF_QP_SHIFT)&
                              GFMX_ENC_CF_QP_MASK;
```

Compressed Bit Stream Pseudocode (continued)

```
NvU8 newPacket = MBInfo & GFMX_ENC_CF_NEW_PACKET;
if (MBInfo & GFMX_ENC_CF_INTER)
{// Inter MB.
    if (MBInfo & (GFMX_ENC_CF_CR_CODED|
                  GFMX_ENC_CF_CB_CODED|
                  GFMX_ENC_CF_Y3_CODED|
                  GFMX_ENC_CF_Y2_CODED|
                  GFMX_ENC_CF_Y1_CODED|
                  GFMX_ENC_CF_Y0_CODED|
                  GFMX_ENC_CF_MV_CODED) )
    {// Not coded. Do not need further parsing for this MB.
        return;
    }
    NvU16 numofMVs = 1;
    if (MBInfo & GFMX_ENC_CF_4MV_CODED)
    {// inter4v.
        numofMVs = 4;
    }
    else
    {
        if (QPStep != previousQPStep)
        {// inter+q.
        }
        else
        {// inter.
        }
    }
    for (i = 0; i < numofMVs; i++)
    {
        horizontal_mv_data[i] = *pOutput &
                                GFMX_ENC_CF_MV_MASK;
        pOutput++;
        vertical_mv_data[i] = *pOutput &
                              GFMX_ENC_CF_MV_MASK;
        pOutput++;
    }
    NvU16 uiBlockCoded = GFMX_ENC_CF_Y0_CODED;
    For (i = 0; i < 6; i++)
```

Compressed Bit Stream Pseudocode (continued)

```
        {
            if (MBInfo & uiBlockCoded)
                block();
            uiBlockCoded = uiBlockCoded >> 1;
        }
    }
    else
    {// Intra MB.
        if (QPStep != previuosQPStep)
        {// intra+q.
        }
        else
        {// intra.
        }
        NvU16 uiBlockCoded = GFMX_ENC_CF_Y0_CODED;
        For (i = 0; i < 6; i++)
        {
            if (short_video_header)
            {
                dc_coefficient = *((NvS16*)pOutput);
                    // 2's complement.
                pOutput+=2;
            }
            else
            {
                ac_pred_flag = MBInfo &
                               GFMX_ENC_CF_AC_PRED_ENABLE;
                if (use_intra_dc_vlc)
                {
                    dc_differential = *((NvS16*)pOutput);
                        // 2's complement.
                    pOutput+=2;
                }
            }
            if (MBInfo & uiBlockCoded)
                block();
            uiBlockCoded = uiBlockCoded >> 1;
        }
```

Compressed Bit Stream Pseudocode (continued)

```
    }
}

block()
{
    NvU16 last = 0;
    do
    {
        last = (*pOutput) & GFMX_ENC_CF_COEF_LAST;
        run = (*pOutput) & GFMX_ENC_CF_COEF_RUN_MASK;
        if (*pOutput & GFMX_ENC_CF_COEF_TWO_BYTES)
        {
            level = *((NvS16*)pOutput);
                // Two bytes, 2's complement.
            pOutput++;
        }
        else
        {
            level = *((NvS8*)pOutput);
                // One byte, 2's complement.
        }
        pOutput++;
    }while (last==0)
}
```

Rate Control

The GFMxEncAPI implements rate control for two use cases: variable bit rate (VBR) and constant bit rate (CBR). For the VBR case (such as video telephony), the wireless channel bandwidth is usually 64 Kbps. Because of the nature of wireless transmission, the channel may not always be able to provide 64 Kbps to the application—it could be lower. Basically, the wireless channel provides an unstable transfer rate.

An application normally maintains a buffer—**`uiBufferSize`** in **`GFMXENCVOL`**—of 0.5 (or 0.25) seconds to hold a few encoded frames. Let's use `totalUntransferredContentSize` to indicate the total size of the encoded frames in this buffer. Depending on channel conditions, data may be

transferred from the buffer quickly or slowly. If the data transfer is quick, `totalUntransferredContentSize` is rapidly reduced. If there is channel congestion, `totalUntransferredContentSize` is reduced slowly because the data transfer is slow.

Whatever the transfer speed, the application constantly fetches encoded frames from the encoder's buffer. Whenever a new frame is fetched from the encoder, `totalUntransferredContentSize` is increased by the new frame size. This `totalUntransferredContentSize` is returned to the encoder, so it can monitor the transfer rate at runtime and adjust the bit rate. If the encoder finds `totalUntransferredContentSize` is too large relative to **`uiBufferSize`**, it compresses the bit stream more. If the encoder finds `totalUntransferredContentSize` is too small, it generates a larger frame (using less compression).

For the CBR case (such as a camcorder), the transfer bit rate is stable. Because the encoder handles the CBR and VBR cases in the same way, the application must track `totalUntransferredContentSize` based on the elapsed time, not the number of frames, because the frame size varies. In this case, `totalUntransferredContentSize` is actually **`uiUTContentSize`** in **`GFMXENCFETCHVOP`**.

Based on the camera's frame rate and the desired bit rate, the encoder can figure out the ideal sizes for I frames and P frames. (In most situations, the I frame size should be three times the P frame size.) Let's use `suggestedSize` to represent an ideal I or P frame size. When the encoder starts to encode a new frame, it compares **`uiUTContentSize`** with the buffer size, located in **`uiBufferSize`** in **`GFMXENCVOL`**. If **`uiUTContentSize`** is less than half of **`uiBufferSize`**, the encoder encodes the frame so it is larger than `suggestedSize`. The smaller **`uiUTContentSize`** is in relation to half of **`uiBufferSize`**, the larger the frame will be. If **`uiUTContentSize`** is larger than half of **`uiBufferSize`**, the encoder makes the frame smaller than `suggestedSize`. The larger **`uiUTContentSize`** is in relation to half of **`uiBufferSize`**, the smaller the frame will be. The bottom line is that the encoder tries to keep the buffer half full.

After the encoder determines the size of the new frame, it starts encoding it macroblock by macroblock. The encoder closely monitors the size of each macroblock. If a macroblock is too large, the encoder uses a larger QP value to compress the next macroblock more. If a macroblock is too small, the encoder uses a smaller QP value to compress the next macroblock less. These dynamic

QP values are limited by **uiMaxIQP**, **uiMinIQP**, **uiMaxPQP**, and **uiMinPQP** in **GFMXENCVOP**. For example, if the encoder decides to increase the I frame QP value but finds the new QP value is greater than **uiMaxIQP**, the encoder uses **uiMaxIQP** instead. This may force the encoder to generate a bigger frame than it originally planned on. If this keeps happening, the encoder eventually decides to drop certain frames to maintain the bit rate. The application normally uses this approach to get better image quality, but the frame rate may drop.

MPEG-4 Encoder Interrupt API

The MPEG-4 encoder interrupt is a component-level interrupt that is supported through the GFMxEncAPI. The MPEG-4 encoder module produces a frame-encoding-done interrupt when it is enabled. An application using an MPEG-4 hardware encoder can take advantage of frame-encoding-done interrupt support to free the CPU from polling for the MPEG-4 encoder status and thus boost system performance. Four functions have been provided that enable, disable, clear, and handle the MPEG-4 encoder module interrupt.

Note: For additional information about the GoForce interrupt architecture, see "Interrupt Architecture API (GFINTxAPI)" on page 109.

MPEG-4 Encoder Interrupt API Service Routines

The MPEG-4 encoder interrupt API consists of four service routines and an associated data type:

GFMxEncInterruptEnable()

This function turns on the MPEG-4 encoder module-level interrupt. The parameter **MxHandle** is obtained by using the following code:

```
id.ComponentType = GF_MXEAPI;
id.DeviceID = GF_DEVICE_ID_DEFAULT;
id.DeviceRev = GF_DEVICE_REV_DEFAULT;
status = GFRmComponentGet( RmHandle, &id, &MxHandle,
                           GF_STATE_DEFAULT );
```

The same code can be used for the other three functions in this section.

Function Prototype

```
GF_RETTYPE  GFMxEncInterruptEnable(
    GF_HANDLE     MxHandle,
    MXENCINTTYPE  IntType );
```

Parameters

MxHandle	The handle from "GFRmComponentGet()" on page 26.
IntType	Module-level interrupt type. It is defined in "MXENCINTTYPE" on page 446.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncInterruptDisable()

This function disables the MPEG-4 encoder module-level interrupt.

Function Prototype

```
GF_RETTYPE  GFMxEncInterruptDisable(
    GF_HANDLE     MxHandle,
    MXENCINTTYPE  IntType );
```

Parameters

MxHandle	The handle from "GFRmComponentGet()" on page 26. See also the description of "GFMxEncInterruptEnable()" on page 444.
IntType	Module-level interrupt type. It is defined in "MXENCINTTYPE" on page 446.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncInterruptClear()

This function clears the MPEG-4 encoder module-level interrupt.

Function Prototype

```
GF_RETTYPE  GFMxEncInterruptClear(
    GF_HANDLE     MxHandle,
    MXENCINTTYPE  IntType );
```

Parameters

MxHandle	The handle from "GFRmComponentGet()" on page 26. See also the description of "GFMxEncInterruptEnable()" on page 444.
IntType	Module-level interrupt type. It is defined in "MXENCINTTYPE" on page 446.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxEncInterruptHandler()

This function performs the MPEG-4 encoder module-level interrupt task. An application should call this function to fetch the encoded VOP bit stream after the MPEG-4 frame-encoding-done interrupt.

Function Prototype

```
GF_RETTYPE  GFMxEncInterruptHandler(
    GF_HANDLE          MxHandle,
    MXENCINTTYPE       IntType,
    PGFMXENCFETCHVOP   pFetchVOP );
```

Parameters

MxHandle	The handle from "GFRmComponentGet()" on page 26. See also the description of "GFMxEncInterruptEnable()" on page 444.

Parameters (continued)

IntType	Module-level interrupt type. It is defined in "MXENCINTTYPE".
pFetchVOP	Structure to hold the encoded VOP information. This structure is defined in "GFMXENCFETCHVOP" on page 433.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

MXENCINTTYPE

The **MXENCINTTYPE** enumerated data type has only one value.

MXENCINTTYPE

```
typedef enum {
    MXENC_FRAME_ENCODE_DONE,
}  MXENCINTTYPE;
```

MPEG-4 Encoder Interrupt Programming Assistance

1. Enable the MPEG-4 module-level interrupt.

```
GFMxEncInterruptEnable( MxHandle, XENC_FRAME_ENCODE_DONE );
```

2. Enable the system-level interrupt.

```
GFINTxInitialize( INTxHandle,SYSINTR_NV_MEI,hEventMEI,NULL,0 );
```

3. The MPEG-4 module-level interrupt handler does the work of interrupt handling. It may be necessary to call this function many times to complete the task.

```
GFMxEncInterruptHandler( MxHandle, MXENC_FRAME_ENCODE_DONE,
                         &fVOP );
```

4. Clear the MPEG-4 module-level interrupt.

```
GFMxEncInterruptClear( MxHandle, MXENC_FRAME_ENCODE_DONE );
```

5. The system-level interrupt is done.

```
GFINTxDone( INTxHandle, SYSINTR_NV_MEI );
```

6. Disable the system-level interrupt.

```
GFINTxDisable( INTxHandle, SYSINTR_NV_MEI );
```

CHAPTER

13

Low-Level MPEG-4 Decoder API (GFMxDecAPI)

GFMxDecAPI Reference

The MPEG-4 decoder API (GFMxDecAPI) consists of the functions described under "GFMxDecAPI Functions" on page 447, and the data structures described under "GFMxDecAPI Data Structures" on page 456.

GFMxDecAPI Functions

The GFMxDecAPI functions include the following:

- "GFMxDecInterruptHandler()" on page 455

GFMxDecGetProperty()

This function returns a variety of information, including the version of the GFMxDecAPI module. It is a good practice to call this function to query for the GFMxDecAPI version and its capabilities before proceeding to other GFMxDecAPI functions.

Function Prototype

```
GF_RETTYPE  GFMxDecGetProperty(
    GF_HANDLE    MXhandle,
    PGFPROPERTY  pMXProp );
```

Parameters

MXhandle	Handle specific to the GFMxDecAPI.
pMXProp	Pointer to a GFPROPERTY structure.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecGetStatus()

This function returns the hardware decoder's status.

Function Prototype

```
GF_RETTYPE  GFMxDecGetStatus(
    GF_HANDLE  Mxhandle,
    NvU32      *pStatus );
```

Parameters

MXhandle	Handle specific to the GFMxDecAPI.
*pStatus	See "GFMxDecGetStatus Definitions".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecGetStatus Definitions

```
/*
    pStatus
*/
#define GF_DECODER_IDLE                         0x00000001
    // If hardware decoder is idle.
#define GF_DECODE_BUSY                          0x00000002
    // If hardware decoder is busy.
#define GF_PP_IDLE                              0x00000004
    // If the postprocessing engine is idle.
#define GF_PP_BUSY                              0x00000008
    // If the postprocessing engine is busy.
```

GFMxDecSetVOP()

This function sets the characteristics of a new VOP. This function should be called on a VOP-by-VOP basis. Please refer to "Programming Assistance" on page 464 for details.

Function Prototype

```
GF_RETTYPE  GFMxDecSetVOP(
    GF_HANDLE    MXhandle,
    PGFMXDECVOP  pVOP );
```

Parameters

MXhandle	Handle specific to the GFMxDecAPI.
pVOP	Set new VOP information.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecSetMBs()

This function decodes the macroblock information passed by the decoder application. It does Inverse Scan, Inverse DC&AC Prediction, Inverse Quantization, Inverse DCT, and Motion Compensation. The decoder

application can call this function and pass as little as one macroblock of information (with more overhead) or pass all the macroblocks in the VOP.

Function Prototype

```
GF_RETTYPE  GFMxDecSetMBs(
    GF_HANDLE   MXhandle,
    PGFMXDECMB  pMBs,
    NvU32       uiNumofMBs );
```

Parameters

MXhandle	Handle specific to the GFMxDecAPI.
pMBs	Pointer to a buffer of GFMXDECMB.
uiNumofMBs	Number of GFMXDECMB in the pMBs buffer. The minimum is 1, the maximum is the number of macroblocks in one VOP.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecPostProcessing()

This function applies the postprocessing filters to the source surface, and outputs the new image to the destination surface.

When the **MXDEC_PP_AUTO** flag is on, this function prepares to do postprocessing only. After **GFMxDecSetMBs()** finishes, the GFMxDecAPI triggers postprocessing automatically. The decoder application does not need to call this function every frame. The application needs to set the **numofDestSurf** and **numofSrcSurf** fields.

When the **MXDEC_PP_AUTO** flag is off, the decoder application can manipulate the postprocessing frame by frame.

If the **MXDEC_PP_RELOAD_QUANTIZER** flag is set, the decoder application should set **lpQuantizers** to the source VOP's quantizers. The **MXDEC_PP_AUTO** flag should be off.

When **MXDEC_PP_RELOAD_QUANTIZER** is off, the GFMxDecAPI uses the latest decoded VOP's quantizers for postprocessing.

Function Prototype

```
GF_RETTYPE  GFMxDecPostProcessing(
    GF_HANDLE   MXhandle,
    PGFMXDECPP  pP );
```

Parameters

MXhandle	Handle specific to the GFMxDecAPI.
pP	Pointer to GFMXDECPP.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecSetAttribute()

This function sets the GFMxDec API options.

Function Prototype

```
GF_RETTYPE GFMxDecSetAttribute(
     GF_HANDLE  Mxhandle,
     NvU32      uiFeature,
     NvU32    * pInfo);
```

Parameters

MxHandle	Handle specific to the GFMxDecAPI.
uiFeature	See "GFMxDecSetAttribute() Definitions".
*pInfo	Pointer to the information buffer. See "GFMxDecSetAttribute() Definitions".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecSetAttribute() Definitions

```
#define MXDEC_ATTR_PRE_DETERMINED_BOUNDARY          0x00000001
```

GFMxDecSetAttribute() Definitions (continued)

```
     //The application determines the decoding boundary (VOL
       and VOP boundary), for MPEG4. For certain file formats,
       the boundary information is stored in the video file,
       and the application can easily use the information to
       determine the boundary.
     // If the application wants the API to detect the
        boundary, the application should not disable this
        attribute. This is primarily for the streaming case,
        when the application does not have the boundary
        information handly. The API automatically detects the
        boundaries. The application does not need to parse the
        bitstream to detect the boundaries.
    //*pInfo: 1 - The application determines the decoding
                  boundary
    //*pInfo: 0 - The application does not determine the
                  decoding boundary
```

GFMxDecGetAttribute()

This function gets the GFMxDec API options.

Function Prototype

```
GF_RETTYPE GFMxDecGetAttribute(
     GF_HANDLE  Mxhandle,
     NvU32      uiFeature,
     NvU32     *pInfo);
```

Parameters

MxHandle	Handle specific to the GFMxDecAPI.
uiFeature	See "GFMxDecGetAttribute() Definitions".
*pInfo	Pointer to the information buffer. See "GFMxDecGetAttribute() Definitions".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecGetAttribute() Definitions

```
#define MXDEC_ATTR_PRE_DETERMINED_BOUNDARY          0x00000001
```

GFMxDecGetAttribute() Definitions (continued)

```
     //The application determines the decoding boundary (VOL
       and VOP boundary), for MPEG4. For certain file formats,
       the boundary information is stored in the video file,
       and the application can easily use the information to
       determine the boundary.
     // If the application wants the API to detect the
        boundary, the application should not disable this
        attribute. This is primarily for the streaming case,
        when the application does not have the boundary
        information handly. The API automatically detects the
        boundaries. The application does not need to parse the
        bitstream to detect the boundaries.
    //*pInfo: 1 - The application determines the decoding
                  boundary
    //*pInfo: 0 - The application does not determine the
                  decoding boundary

#define MXDEC_ATTR_MP4_DEC_VOP_RECT_TOP_ALIGNMENT   0x00000002
     //If the application wants to set pDestRect in GFMXDECVOP
       or GFMXDECMP4DECVOP, the application must use this
       attribute to get allignment information and align the
       rectangle top & left corner accordingly.
     //*pInfo: Top field alignment in term of lines

#define MXDEC_ATTR_MP4_DEC_VOP_RECT_LEFT_ALIGNMENT  0x00000003
     //If the application wants to set pDestRect in GFMXDECVOP
       or GFMXDECMP4DECVOP, the application must use those
       attributes to get alignment information and align the
       rectangle top & left corner accordingly.
    //*pInfo: Left field alignment in term of pixels.
```

GFMxDecSet()

This function sets up the GFMxDecAPI callback function.

Function Prototype

```
GF_RETTYPE GFMxDecSet(
     GF_HANDLE  MXhandle,
     NvU32      uiFeature,
     void      *pInfo);
```

Parameters

MxHandle	Handle specific to the GFMxDecAPI.
uiFeature	See "GFMxDecSet() Definitions".
*pInfo	Set up decoder option: Pointer to "GFMXDECCALLBACK".

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecSet() Definitions

```
#define MXDEC_SET_READBITSTREAM         1
      // Set up READBITSTREAM callback function
```

GFMxDecInterruptControl()

This function provides component-level interrupt control for the GFMxDecAPI.

Function Prototype

```
GF_RETTYPE GFMxDecInterruptControl (
    GF_HANDLE                           MxHandle,
    GFMX_DEC_INTERRUPT_TYPE             IntType,
    GFMX_DEC_INTERRUPT_OPERATION_TYPE   op,
    void                                *pData );
```

Parameters

MXhandle	Handle specific to the GFMxDecAPI.
IntType	GFMxDecAPI interrupt type as defined in "GFMX_DEC_INTERRUPT_TYPE" on page 463.
op	GFMxDecAPI interrupt operation as defined in "GFMX_DEC_INTERRUPT_OPERATION_TYPE" on page 462.
pData	Pointer to data being passed in or out.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecInterruptHandler()

This function provides component-level interrupt control for the GFMxDecAPI. It is optional because an interrupt service thread (IST) at the application level can call the GFMxDecAPI to complete the task.

Function Prototype

```
GF_RETTYPE GFMxDecInterruptHandler (
    GF_HANDLE                MxHandle,
    GFMX_DEC_INTERRUPT_TYPE  IntType,
    void                     *pData );
```

Parameters

MXhandle	Handle specific to the GFMxDecAPI.
IntType	GFMxDecAPI interrupt type as defined in "GFMX_DEC_INTERRUPT_TYPE" on page 463.
pData	Pointer to data being passed in or out.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFMxDecAPI Data Structures

GFPROPERTY

This structure is described in "GFPROPERTY" on page 10. The GFMxDecAPI **`GFPROPERTY`** capabilities are defined below.

GFPROPERTY Definitions

```
/*
    Capability Definitions
*/
#define GF_MXDEC_CAP_SIMPLE                 0x00000001
    // Support simple profile.
#define GF_MXDEC_CAP_SCALE                  0x00000002
    // Support simple scale profile.
#define GF_MXDEC_CAP_CORE                   0x00000004
    // Support CORE profile.
#define GF_MXDEC_CAP_MAIN                   0x00000008
    // Support MAIN profile.
#define GF_MXDEC_CAP_N_BIT                  0x00000010
    // Support N_bit profile.
#define GF_MXDEC_CAP_LEVEL1                 0x00000100
    // Support LEVEL 1.
#define GF_MXDEC_CAP_LEVEL2                 0x00000200
    // Support LEVEL 2.
#define GF_MXDEC_CAP_LEVEL3                 0x00000400
    // Support LEVEL 3.
#define GF_MXDEC_CAP_LEVEL4                 0x00000800
    // Support LEVEL 4.
```

GFPROPERTY Definitions (continued)

```
#define GF_MXDEC_CAP_DB                         0x00001000
    // Support post-processing de-blocking.
#define GF_MXDEC_CAP_DR                         0x00002000
    // Support post-processing de-ringing.
```

GFMXDECVECTOR

This structure holds motion vector information.

GFMXDECVECTOR Structure

```
typedef struct  _GFMXDECVECTOR {
    NvS8  H_MV_DATA;
    NvU8  H_MV_RES;
    NvS8  V_MV_DATA;
    NvU8  V_MV_RES;
} GFMXDECVECTOR;
```

GFMXDECVECTOR Fields

H_MV_DATA	horizontal_mv_data
H_MV_RES	horizontal_mv_residual
V_MV_DATA	vertical_mv_data
V_MV_RES	vertical_mv_residual

GFMXDECCOEF

The contents in **GFMXDECCOEF** are the coded coefficient values that the decoder application decoded from the bit stream.

GFMXDECCOEF Structure

```
typedef struct  _GFMXDECCOEF {
    NvU8   uiLAST;
    NvU8   uiRUN;
    NvS16  iLEVEL;
} GFMXDECCOEF;
```

GFMXDECCOEF Fields

uiLAST	This last coefficient in current block.

GFMXDECCOEF Fields (continued)

uiRUN	How many zeroes precede this coefficient.
iLEVEL	This coefficient's level value.

GFMXDECMB

This structure holds the macroblock information.

GFMXDECMB Structure

```
typedef struct  _GFMXDECMB {
    NvU16          uiX;
    NvU16          uiY;
    NvU16          uiMBInfo;
    NvU16          uiQuantiser;
    GFMXDECVECTOR  Vector[4];
    NvU32          uiNumofCoef;
} GFMXDECMB;
```

GFMXDECMB Fields

uiX	Macroblock's x position in the frame
uiY	Macroblock's y position in the frame
uiMBInfo	Macroblock's characteristics
uiQuantiser	Current macroblock's quantiser
Vector[4]	Motion vector zero to three
uiNumofCoef	Number of GFMXDECCOEF structures for current macroblock. If uiNumofCoef is not zero, GFMXDECCOEF structure follows after the GFMXDECMB.

GFMXDECMB Definitions

```
/*
    uiMBInfo
*/
#define MXDEC_MB_TYPE_INTER                              0x1
    // Inter macro block.
#define MXDEC_MB_CR_BLOCK                                0x2
    // cr block coded.
#define MXDEC_MB_CB_BLOCK                                0x4
    // cb block coded.
#define MXDEC_MB_Y3_BLOCK                                0x8
    // y3 block coded.
```

GFMXDECMB Definitions (continued)

```
#define MXDEC_MB_Y2_BLOCK                              0x10
    // y2 block coded.
#define MXDEC_MB_Y1_BLOCK                              0x20
    // y1 block coded.
#define MXDEC_MB_Y0_BLOCK                              0x40
    // y0 block coded.
#define MXDEC_MB_MV_4_MVS                              0x80
    // 4-motion vector coded.
#define MXDEC_MB_NOT_CODED                            0x100
    // Macroblock not coded.
#define MXDEC_MB_AC_PRED_ON                           0x200
    // AC prediction is on.
#define MXDEC_MB_IN_NEW_PACKET                        0x400
#define MXDEC_MB_DC_CODED_AS_AC                       0x800
#define MXDEC_MB_AUTO_PP                             0x1000
    // If GFMxDecPostProcessing() is called with the
    // MXDEC_PP_AUTO flag on, the GFMxDecAPI decodes this
    // macroblock and auto-triggers postprocessing.
#define MXDEC_MB_AUTO_DISPLAY                        0x2000
    // After GFMxDecAPI decodes this macroblock, it auto-
    // triggers the video engine to display this VOP if
    // the application set auto-blit on with GFVxBlt().
    // If MXDEC_MB_AUTO_PP and MXDEC_MA_AUTO_DISPLAY are
    // both on, GFMxDecAPI triggers postprocessing and then
    // triggers the display engine.
```

GFMXDECVOP

This structure holds the video object plane (VOP) information.

GFMXDECVOP Structure

```
typedef struct  _GFMXDECVOP {
    NvU32         uiVOPinfo;
    NvU32         uiFFORWARD;
    PGFRMSURFACE  pRef;
    PGFRMSURFACE  pCur;
    PGFRECT       pRefSurfRect;
    PGFRECT       pCurSurfRect;
}  GFMXDECVOP;
```

GFMXDECVOP Fields

uiVOPinfo	Set new VOP information.
uiFFORWARD	FCODE_FORWARD. Ignore this value if it is an I frame.
pRef	Reference VOP. Ignore this item if current VOP is an I.
pCur	Current VOP.
pRefSurfRect	Specifies a rectangular area within the pRef surface for the GFMxDecAPI to use with the reference image. This field only takes effect when MXDEC_VOP_SPECIFY_REFSURF_RECT is set. The rectangle's top and left should be aligned by the application, which should call GFMxDecGetAttribute() with MXDEC_ATTR_MP4_DEC_VOP_RECT_TOP_ALIGNMENT and MXDEC_ATTR_MP4_DEC_VOP_RECT_LEFT_ALIGNMENT to get the required alignment. The rectangle's width and height should exactly match the image width and height that is encoded in the bit stream.
pCurSurfRect	Specifies a rectangular area of the pCur surface within which the GFMxDecAPI should place the decoded image output. This field only takes effect if MXDEC_VOP_SPECIFY_CURSURF_RECT is set. The rectangle's top and left should be aligned by the application, which should call GFMxDecGetAttribute() with MXDEC_ATTR_MP4_DEC_VOP_RECT_TOP_ALIGNMENT and MXDEC_ATTR_MP4_DEC_VOP_RECT_LEFT_ALIGNMENT to get the required alignment. The rectangle's width and height should exactly match the image width and height that is encoded in the bit stream.

GFMXDECVOP Definitions

```
/*
    uiVOPFlag32 Values
*/
#define MXDEC_VOP_P_VOP                                0x1
    // Indicates a P VOP. If not set, indicates an I VOP.
#define MXDEC_VOP_ROUNDING_ONE                         0x2
    // VOP_ROUNDING_TYPE is one.
```

GFMXDECVOP Definitions (continued)

```
#define MXDEC_VOP_SHORT_HEADER                          0x4
    // This bit stream is in short video header format.
#define MXDEC_VOP_SPECIFY_REFSURF_RECT                  0x8
    // If this flag is set, pRefSurfRect must be set, and
    // GFMxDecAPI references the image in the pRefSurfRect
    // area of the pRef surface. Otherwise, GFMxDecAPI
    // references the entire pRef surface area.
#define MXDEC_VOP_SPECIFY_CURSURF_RECT                0x10
    // If this flag is set, pCurSurfRect must be set, and
    // GFMxDecAPI outputs decoded image to the pCurSurfRect
    // area of the pCur surface. Otherwise, GFMxDecAPI
    // outputs to the pCur surface starting from top-left
    // corner with the width and height equal to the pCur
    // surface width and height.
```

GFMXDECPP

This structure holds the postprocessing information.

GFMXDECPP Structure

```
typedef struct  _GFMXDECPP {
    PGFRMSURFACE  *pDestSurf;
    PGFRMSURFACE  *pSrcSurf;
    PGFRECT       pRect;
    NvU32         numofDestSurf;
    NvU32         numofSrcSurf;
    NvU8          *lpQuantiser;
    NvU32         PPOption;
}  GFMXDECPP;
```

GFMXDECPP Fields

*pDestSurf	Pointer to an array of surfaces that holds the result from postprocessing. Pointer to an array of surfaces to accommodate automatic postprocessing
*pSrcSurf	Pointer to an array of surfaces to be postprocessed. Pointer to an array of surfaces to accommodate automatic postprocessing.
pRect	Rectangular area to be processed for both source and destination surfaces. See "GFRECT" on page 11.

GFMXDECPP Fields (continued)

numofDestSurf	If MXDEC_PP_AUTO flag is set, this parameter must be filled.
numofSrcSurf	If MXDEC_PP_AUTO flag is set, this parameter must be filled.
*lpQuantiser	Pointer to array of quantization steps (QPs) for macroblocks in the source VOP. The GFMxDecAPI automatically saved the last two decoded VOP's QP tables. The application may not need to reload the table if it can make sure the source VOP is one of the last decoded VOPs. This field must be filled if MXDEC_PP_RELOAD_QUANTIZER is set.
PPOption	See "GFMXDECPP Definitions" on page 462.

GFMXDECPP Definitions

```
/*
    PPOption
*/
#define MXDEC_PP_DB_ON                                  0x1
    // Turn on deblocking filter for this frame.
#define MXDEC_PP_DR_ON                                  0x2
    // Turn on de-ringing filter for this frame.
#define MXDEC_PP_AUTO                                   0x4
    // Auto-trigger postprocessing by hardware.
#define MXDEC_PP_RELOAD_QUANTIZER                       0x8
    // Reload the source VOP's quantizers.
```

GFMX_DEC_INTERRUPT_OPERATION_TYPE

This data type is used by "GFMxDecInterruptControl()" on page 454.

GFMX_DEC_INTERRUPT_OPERATION_TYPE Enumerated Type

```
typedef enum {
    GFMX_DEC_INTERRUPT_ENABLE,
    GFMX_DEC_INTERRUPT_DISABLE,
    GFMX_DEC_INTERRUPT_CLEAR
} GFMX_DEC_INTERRUPT_OPERATION_TYPE;
```

GFMX_DEC_INTERRUPT_TYPE

This data type is used by "GFMxDecInterruptHandler()" on page 455 and "GFMxDecInterruptControl()" on page 454.

GFMX_DEC_INTERRUPT_TYPE Enumerated Type

```
typedef enum {
    GFMX_DEC_POST_PROCESSING_DONE_INTR,
    GFMX_DEC_DATA_STREAM_THRESHOLD_INTR,
    GFMX_DEC_CMD_FIFO_THRESHOLD_INTR,
    GFMX_DEC_COEFFIICIENT_FIFO_THRESHOLD_INTR,
    GFMX_DEC_DECODE_DONE_INTR
}  GFMX_DEC_INTERRUPT_TYPE;
```

GFMXDECCALLBACK

This structure is used by "GFMxDecSet()".

```
typedef struct _GFMXDECCALLBACK
{
    void   *pPara;
    NvU32 (*pCallBack)(void * pPara,
                       NvU8 ** ppBuffer,
                       NvS32 * BufferLength,
                       NvU32 uFlag);
}GFMXDECCALLBACK, *PGFMXDECCALLBACK;
```

GFMXDECCALLBACK Fields

`pPara`	Parameter passed between the application and the API.

`(*pCallBack)`
`(void * pPara,`
`NvU8 ** ppBuffer,`
`NvS32 * BufferLength,`
`NvU32 uFlag);`
	The GFMJxDecAPI uses this callback function to ask the application to pass in the bitstream.
	If the application sets **`MXDEC_ATTR_PRE_DETERMINED_BOUNDARY`**, then the application should return **`MXDEC_BOUNDARY_REACHED`** when the boundary is detected.
	If the application does not set **`MXDEC_ATTR_PRE_DETERMINED_BOUNDARY`**, 0 should be returned.

Programming Assistance

The following sections are provided to show how to use the GFMxDecAPI effectively.

GFMxDecAPI Programming Sequence

The following procedure requires that **`GFRmOpen()`** is called first to start GFSDK usage.

1. The decoder application should call **`GFRmComponentGet()`** with **`GF_VXAPI`** to obtain a GFVxAPI **`VXhandle`** first.
2. Query the properties with **`GFVxGetProperty()`** to see whether this GFVxAPI version supports MPEG decoding.
3. If it supports MPEG decoding, call **`GFRmComponentGet()`** again with **`GF_MXDAPI`** to obtain a GFMxDecAPI **`MXhandle`**.
4. Call **`GFMxDecGetProperty()`** to query properties. Check whether this GFMxDAPI version can support the desired MPEG profile and level.
5. If the properties check out, call **`GFRmSurfaceAlloc()`** to allocate at least one reference surface and one current surface for decoding purposes.
6. If the postprocessing engine is needed, allocate one surface to hold the postprocessing results.
7. For display purposes, the decoder application could call **`GFVxBlt()`** for the following two scenarios. Please refer to the latest GFVxAPI document for additional information.
   - Performing color space conversion and then a stretch blit to the primary surface directly.

- Blitting to the overlay surface.

8. The decoder application should decode VOP-by-VOP. Call **`GFMxDecSetVOP()`** to set up the VOP's characteristics.
9. After the decoder application has decoded enough macroblocks, call **`GFMxDecSetMBs()`** to trigger the hardware to decode those macroblocks until all macroblocks in the current VOP are finished. **`GFMxDecSetMBs()`** may initiate the postprocessing engine or video engine automatically.
10. Before exiting the application, the decoder application should call **`GFRmSurfaceFree()`** to free the surfaces that have been allocated.
11. Call **`GFRmComponentRelease()`** with the **`MXhandle`** to release the resources of the decoder module.
12. Lastly, call **`GFRmComponentRelease()`** with the **`VXhandle`** to release the resources of the video engine.

Programming uiVOPInfo in GFMXDECVOP

The pseudocode below refers to "GFMXDECVOP" on page 459.

uiVOPInfo Pseudocode

```
If (short_video_header)
    uiVOPInfo = MPEGDEC_VOP_SHORT_HEADER;
else
    uiVOPInfo = 0;

if (vop_coding_type == ”P”) {
    uiVOPInfo |= MPEGDEC_VOP_P_VOP;
    if (vop_rounding_type)
        uiVOPInfo |= MPEGDEC_VOP_ROUNDING_ONE;
}
```

Handling Uncoded VOP

The decoder application does not need do anything. The display screen keeps displaying the last VOP's image.

Programming GFMXDECMB

Please refer to "GFMXDECMB" on page 458 for the context for the discussions in this section.

- **uiX, uiY**

  This is the current macroblock's position in the VOP. For example, the very top-left macroblock's position is **uiX** = 0, **uiY** = 0. The second one immediately following the first one is **uiX** = 1, **uiY** = 0. The one immediately below the first macroblock is **uiX** = 0, **uiY** = 1. The maximum value for **uiX** or **uiY** is 63.

- **uiMBInfo**
  - If the current macroblock is not coded, the decoder application should set the **MXDEC_MB_NOT_CODED** flag (to on).
  - If the current macroblock is an Inter macroblock, set the flag **MXDEC_MB_TYPE_INTER**. If any block is coded, set the corresponding flag.
  - If there are four motion vectors coded instead of one, the decoder application should set the **MXDEC_MB_MV_4_MVS** flag.
  - If AC prediction should be on for the current macroblock, set the **MXDEC_MB_AC_PRED_ON** flag.
  - If the current macroblock is the first macroblock in the packet or GOB, the decoder application should set the **MXDEC_MB_IN_NEW_PACKET** flag.
  - If DC is coded as AC, set the **MXDEC_MB_DC_CODED_AS_AC** flag.
  - If the decoder application wants the GFMxDecAPI to do postprocessing automatically and to display after this macroblock, it should set the **MXDEC_MB_AUTO_PP** and **MXDEC_MB_AUTO_DISPLAY** flags. Please refer to how to do postprocessing automatically in this document and how to display automatically in the GFVxAPI.

- **uiQuantiser**

  This field is **quant_scale** from the bit stream. If the current macroblock is InterQ or IntraQ, the decoder application should calculate the new quantization scale based on **dQuant** from the bit stream.

- **Vectors**

  If this is an Intra macroblock, ignore those fields. For an Inter macroblock, if only one motion vector is coded, just program **Vector0** and ignore the other three. The decoder application can get the **horizontal_mv_data**,

horizontal_mv_residual, **vertical_mv_data**, and **vertical_mv_residual** from the bit stream. Please refer to the MPEG-4 specification.

- **uiNumofCoef**

  This is the number of **GFMXDECCOEF** structures following the **GFMXDECMB** structure.

Handling Coefficients

If there are any coefficient values in the current macroblock, put them following the **GFMXDECMB** structure. The sequence is DC first, followed by AC value. Block by block, from Y0, Y1, Y2, Y3, Cb to Cr. The decoder application should not do Inverse Scan. Please refer to "Programming GFMXDECCOEF" on page 467.

Handling Skipped Macroblocks

Set **uiMBInfo** to **MXDEC_MB_NOT_CODED**. Set **uiNumofCoef** to 0. Set the **MXDEC_MB_IN_NEW_PACKET** flag if it is the first macroblock in the packet or a GOB.

Programming GFMXDECCOEF

The contents in "GFMXDECCOEF" on page 457 are the coded coefficient values that the decoder application decoded from the bit stream. The decoder application can get values for **LAST**, **RUN**, and **LEVEL**. If **use_intra_dc_vlc** is true, **uiLAST** and **uiRUN** should be zero for a DC coefficient.

GFMXDECCOEF Sample

```
GFMXDECCOEF *pTmp;
If (s == 1) // If current coefficient is negative
    LEVEL = -LEVEL; // Change it to two's complement format.
If (LAST)
    pTmp->uiLAST = 1;
else
     pTmp->uiLAST = 0;
pTmp->uiRUN = RUN;
pTmp->iLEVEL = LEVEL;
```

Handling Missing Macroblocks

For various reasons, the decoder application may not recover certain macroblocks (macroblocks are missing). Possible solutions are as follows:

- For an I frame, the application could duplicate the previous good macroblock.
- For a P frame, the application could treat it as a skipped macroblock. (See "Handling Skipped Macroblocks" on page 467.)

The decoder can come up any creative method to handle missing macroblocks, but the application always needs to pass macroblocks from left to right and from top to bottom to the GFMxDecAPI. The GFMxDecAPI always expects the decoder application to deliver the full number of macroblocks in a VOP.

Automatic Postprocessing

Call **`GFMxDecPostProcessing()`** with the **`MXDEC_PP_AUTO`** flag on. The decoder application must specify which postprocessing filter should be on also. When sending the last macroblock information in through **`GFMxDecSetMBs()`**, the decoder application should set the **`MXDEC_MB_AUTO_PP`** flag on. The GFMxDecAPI then automatically triggers the postprocessing.

CHAPTER

14

H.264/AVC Encoder API (GFMxEncH264API)

GFMxEncH264 API Overview

The H.264/AVC encoder API generates H.264/AVC baseline-compliant bit streams, but has been defined to handle all of the three profiles for future expansion.

GFMxEncH264 API Reference

The H.264/AVC encoder API generates (GFMxEncH264API) consists of the functions described under “GFMxEncH264API Functions” on page 470, and the data structures described under “GFMxEncH264API Data Structures” on page 475.

GFMxEncH264API Functions

The GFMxEndH264API functions include the following:

GFMxEncH264GetProperty()

This API can be used to query H.264 capability.

Function Prototype

```
GF_RETTYPE GFMxEncH264GetProperty (
GF_HANDLE   MXhandle,
PGFPROPERTY pMXProp);
```

Parameters

`MXhandle` The handle specific to GFMxEncH264 API

`pMXProp` Pointer to GFPROPERTY

Return Values

`GF_SUCCESS` If successful

`GF_ERROR` If error

GFMxEncH264SetSequence()

This API can be used to generate H.264/AVC bit streams. It sets up the sequence level characteristics.

Function Prototype

```
GF_RETTYPE GFMxEncH264SetSequence (
GF_HANDLE MXhandle,
PGFMXENCH264SETSEQUENCE pSequence);
```

Parameters

MXhandle	The handle specific to GFMxEnc API
pSequence	Pointer to GFMXENCH264SETSEQUENCE

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFMxEncH264SetPicture()

This API sets the characteristics of a new picture. GFMxEncH264SetPicture should be called after **GFMxEncH264SetSequence()** and before **GFMxEncH264Start()**. If the application decides to change any picture information during the encoding time (after **GFMxEncH264Start()**), the application can call this function to pass the new picture information to the GFMxEncAPI. Otherwise, the application should not call this function, allowing the GFMxEncAPI to reduce overhead.

Function Prototype

```
GF_RETTYPE GFMxEncH264SetPicture (
GF_HANDLE MXhandle,
PGFMXENCH264SETPICTURE pPicture);
```

Parameters

MXhandle	The handle specific to GFMxEnc API
pPicture	Pointer to GFMXENCH264SETPICTURE

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFMxEncH264FeedImage()

The application can call this function if the application has the source image already stored somewhere.

This function should not be called if the image comes directly from the video input port (VIP)— for example, from a camera. If application has the source image and does not want to control the time stamp, the application should not call this API but instead call GFVxFeedImage. The time stamp will be generated by the encoder based on when the application calls GFVxFeedImage.

Function Prototype

```
GF_RETTYPE GFMxEncH264FeedImage (
GF_HANDLE              MXhandle,
PGFMXENCH264FEEDIMAGE pFeedImage );
);
```

Parameters

`MXhandle`	The handle specific to GFMxEnc API
`pFeedImage`	Pointer to GFMXENCH264FEEDIMAGE

Return Values

`GF_SUCCESS`	If successful
`GF_ERROR`	If error

GFMxEncH264FetchNALs()

An application should call this function to fetch the encoded bitstream. If the source image is from the host, the application should call this function after **`GFMxEncH264FeedImage()`**. If the source image is from a video camera connected to the VIP and the application is using a polling scheme, the application should call this function at least as frequently as the camera's frame rate.

If GFMxEnc has several NAL units encoded, **GFMxEncH264FetchNALs** will return back as many bits as can fit in pBuf and pNALLengthBuf (in GFMXENCH264FETCHNALS). Within each **GFMxEncH264FetchNALs** call,

GFMxEncH264FetchNALs will not mix the current access unit with the next access unit. At most, **GFMxEncH264FetchNALs** can return one access unit.

Function Prototype

```
GF_RETTYPE GFMxEncH264FetchNALs(
GF_HANDLE MXhandle,
PGFMXENCH264FETCHNALS pFetchNALs );
```

Parameters

MXhandle	The handle specific to GFMxEnc API
pFetchNALs	Structure to hold encoded bit stream information

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFMxEncH264Start()

This function starts the GFMxEncAPI module to encode the bitstream.

Function Prototype

```
GF_RETTYPE GFMxEncH264Start(
GF_HANDLE MXhandle );
```

Parameters

MXhandle	The handle specific to GFMxEnc API

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFMxEncH264Pause()

This function pauses the encoding process until **`GFMxEncH264Start()`** is called to restart encoding, or **`GFMxEncH264Stop()`** is called to completely stop encoding.

Function Prototype

```
GF_RETTYPE GFMxEncH264Pause(
GF_HANDLE MXhandle );
```

Parameters

`MXhandle` The handle specific to GFMxEnc API

Return Values

`GF_SUCCESS` If successful

`GF_ERROR` If error

GFMxEncH264Stop()

This function stops the encoding process.

Function Prototype

```
GF_RETTYPE GFMxEncH264Stop(
GF_HANDLE  MXhandle );
```

Parameters

`MXhandle` The handle specific to GFMxEnc API

Return Values

`GF_SUCCESS` If successful

`GF_ERROR` If error

GFMxEncH264API Data Structures

The GFMxEncH264API data structures include the following:

GFMXENCH264SETSEQUENCE

This structure holds H.264/AVC sequence information.

GFMXENCH264SETSEQUENCE Structure

```
typedef struct GFMXENCH264SETSEQUENCE_TAG {
Nvu32           SequenceInfo;
NvU32           profile_idc;
NvU32           level_idc;
NvU32           seq_parameter_set_id;
NvU32           log2_max_frame_num_minus_4;
NvU32           pic_order_cnt_type;
NvU32           log2_max_pic_order_cnt_lsb_minus_4;
                //only available when pic_order_cnt_type is 0
NvU32           offset_for_non_ref_pic;
                //only available when pic_order_cnt_type is 1
NvU32           offset_for_top_to_bottom_field;
                //only available when pic_order_cnt_type is 1
NvU32           num_ref_frames_in_pic_order_cnt_cycle;
                //only available when pic_order_cnt_type is 1
NvU32           *poffset_for_ref_frame;
                //only available when pic_order_cnt_type is 1
NvU32           num_ref_frames;
PGFRMSURFACE    *pSurfaces;
NvU32           pic_width;
NvU32           pic_height;
GFRECT          CroppingRect;
PGFRMSURFACE    pSrcSurf;
PGFRECT         pSrcRect;
```

```
float           fFrameRate;
NvU32           uiMaxBitRate;
NvU32           uiBufferSize;
} GFMXENCH264SETSEQUENCE;
```

GFMXENCH264SETSEQUENCE Field Definitions

SequenceInfo	
GF_MXENC_H264_SEQUENCE_CONSTRAINT_SET0_FLAG	0x00000001
GF_MXENC_H264_SEQUENCE_CONSTRAINT_SET1_FLAG	0x00000002
GF_MXENC_H264_SEQUENCE_CONSTRAINT_SET2_FLAG	0x00000004
GF_MXENC_H264_SEQUENCE_FRAME_CROPPING_FLAG	0x00000008
If this flag is set, CroppingRect has the cropping rectangle area.	
GF_MXENC_H264_SEQUENCE_DELTA_PIC_ORDER_ALWAYS_ZERO_FLAG	0x00000010
GF_MXENC_H264_SEQUENCE_GAPS_IN_FRAME_NUMBER_ALLOWED_FLAG	0x00000020
GF_MXENC_H264_SEQUENCE_FRMAE_MBS_ONLY_FLAG	0x00000040
GF_MXENC_H264_SEQUENCE_MB_ADAPTIVE_FRAME_FIELD_ FLAG	0x00000080
GF_MXENC_H264_SEQUENCE_DIRECT_8X8_INFERENCE_ FLAG	0x00000100
GF_MXENC_H264_SEQUENCE_BYTE_STREAM_ENABLE	0x00000200
MxEnc API will generate byte stream	
GF_MXENC_H264_SEQUENCE_DATA_PARTITION_ENABLE	0x00000400
GF_MXENC_H264_SEQUENCE_RATE_CONTROL_ENABLE	0x00000800
GF_MXENC_H264_SEQUENCE_AUTO_IMRF_ENABLE	0x00001000
Enable intra- Macro Block refreshing. This mode could run automatically without any application interaction. Applications could change the macroblocks' counter range by calling GFMxEncH264SetPicture, if desired.	
GF_MXENC_H264_SEQUENCE_MC_IMRF_ENABLE	0x00002000
Enable intra-macroblock refreshing with customized counter matrix from the applcation. When this flag is set, applications must call GFMxEncH264SetPicture to set up the intra-macroblock refresh counter matrix.	
GF_MXENC_H264_SEQUENCE_AUTO_ISRF_ENABLE	0x00004000
Enable intra-slice refreshing. This mode could run automatically without any application interaction. Applications could change the intra slices'counter range by calling GFMxEncH264SetPicture, if desired.	

SequenceInfo

GF_MXENC_H264_SEQUENCE_MC_ISRF_ENABLE	0x00008000
Enable intra slice refreshing with customized counter matrix from the application. When this flag is set, applications must call GFMxEncH264SetPicture to set up the intra-slice refresh counter matrix.	
GF_MXENC_H264_SEQUENCE_PREPARE_ROTATION	0x00010000
The application may need to rotate the encoding image during the encoding process.	

pSrcSurf

Surface to be encoded. If the source image is from a camera, set this field to NULL.

pSrcRect

Area of the source image that needs to be encoded.

fFrameRate

Frame rate per second. This is the real input frame rate. For example, if encoding image is from a camera, fFrameRate should be the camera frame rate and not the encoded bit stream's frame rate.

uiMaxBitRate

Kilobits/second.

uiBufferSize

Size in DWORDs. This buffer is created and managed by the application to temporarily hold the encoded bit stream before transferring the bit stream out.

GFMXENCH264SETPICTURE

This structure holds H.264/AVC picture information.

GFMXENCH264SETPICTURE Structure

```
typedef struct GFMXENCH264SETPICTURE_TAG {
Nvu32      PictureInfo;
NvU32      numofSlicesGroup;
NvU32      sliceGroupMapType;
void       * sliceGroupMapInfo;
NvU32      num_of_ref_idx_10_active;
NvU32      num_of_ref_idx_11_active;
NvS32      pic_init_qp;
NvS32      pic_init_qs;
NvS32      chroma_qp_index_offset;
NvU32      uiMaxIDRQP;        // Max QP value for IDR
NvU32      uiMinIDRQP;        // Min QP value for IDR
NvU32      uiMaxNONIDRQP;     // Max QP value for non-IDR
                                 picture
NvU32      uiMinNONIDRQP;     // Min QP value for non-IDR
                                 picture
NvU32      uiNumofPicBetween2IDRs;
NvU32      uiRNumerator;
NvU32      uiRDenominator;
NvU32      uiIMRFIMinValue;
NvU32      uiIMRFIMaxValue;
NvU32      uiIMRFPMinValue;
NvU32      uiIMRFPMaxValue;
NvU32      uiIMRFDefaultCounter;
NvU8       *pIMRFMatrix;
NvU32      uiISRFIMinValue;
NvU32      uiISRFIMaxValue;
NvU32      uiISRFPMinValue;
NvU32      uiISRFPMaxValue;
NvU32      uiISRFDefaultCounter;
NvU8       *pISRFMatrix;
} GFMXENCH264SETPICTURE; //
```

GFMXENCH264SETPICTURE PictureInfo Field Definition

PictureInfo	
GF_MXENC_H264_PICTURE_CHANGE_QP	0x00000001
When the application enables this flag, the application must set **pic_init_qp**, **pic_init_qs** and **chroma_qp_index_offset**.	
GF_MXENC_H264_PICTURE_CHANGE_NUM_OF_REF_IDX	0x00000002
When application enables this flag, application must set **num_of_ref_idx_10_active** and **num_of_ref_idx_11_active**.	
GF_MXENC_H264_PICTURE_REDUCE_FRAME_RATE	0x00000004
Reduce the frame rate by skipping certain frames. Must set **uiRNumerator** and **uiRDenominator**.	
GF_MXENC_H264_PICTURE_ENCODE_IDR_ASAP	0x00000008
Encode IDR as soon as possible.	
GF_MXENC_H264_PICTURE_ADD_RECOVERY_POINT_ASAP	0x00000010
Encoder should generate recovery point as soon as possible.	
GF_MXENC_H264_PICTURE_CHANGE_MIN_MAX_QP	0x00000020
When application set this flag, application must set **uiMaxIDRQP**, **uiMinIDRQP**, **uiMaxNONIDRQP** and **uiMinNONIDRQP**	
GF_MXENC_H264_PICTURE_MORE_IDR_IN_SEQUENCE	0x00000040
The number of NON-IDR picture between two IDR pictures. When this flag is set, **uiNumofPicBetween2IDRs** must be set.	
GF_MXENC_H264_PICTURE_LOAD_IMRF_COUNTER_RANGE	0x00000080
The application changes the counter range by setting this lag and giving the range in uiIMRFIMinValue, **uiIMRFIMaxValue**, **uiIMRFPMinValue**, and **uiIMRFPMaxValue**. Applications can turn on this flag only when **GF_MXENC_H264_SEQUENCE_AUTO_IMRF_ENABLE** is on.	
GF_MXENC_H264_PICTURE_LOAD_IMRF_MATRIX	0x00000100
The application must fill in **uiIMRFDefaultCounter** and **pIMRFMatrix** if this flag is set. The application can set this flag only if **GF_MXENC_H264_PICTURE_MC_IMRF_ENABLE** is on.	
GF_MXENC_H264_PICTURE_LOAD_ISRF_COUNTER_RANGE	0x00000200
The application changes the counter range by setting this flag and giving the range in **uiISRFIMinValue**, **uiISRFIMaxValue**, **uiISRFPMinValue**, and **uiISRFPMaxValue**. Applications can turn on this flag only when **GF_MXENC_H264_SEQUENCE_AUTO_ISRF_ENABLE** is on.	
GF_MXENC_H264_PICTURE_LOAD_ISRF_MATRIX	0x00000400

PictureInfo

The application must fill in **uiISRFDefaultCounter** and **pISRFMatrix** if this flag is set. Applications can set this flag only if **GF_MXENC_H264_PICTURE_MC_ISRF_ENABLE** is on.

GF_MXENC_264_PICTURE_CONSTRAINED_INTRA_PRED_ENABLE 0x00000800

GFMXENCH264FEEDIMAGE

This structure is used by **GFMxEncH264FeedImage()**.

GFMXENCH264FEEDIMAGE Structure

```
typedef struct GFMXENCH264FEEDIMAGE_TAG {
NvU32 uiTime;
PGFRMSURFACE pImgSurf;
} GFMXENCH264FEEDIMAGE;
```

GFMXENCH264FEEDIMAGE Field Definitions

uiTime

In milliseconds. First picture may start from 0.

pImgSurf

Image to be encoded.

GFMxEncH264FeedImage can support the
GF_SURFACE_YUV420,
GF_SURFACE_YUYV,
GF_SURFACE_YVYU,
GF_SURFACE_UYVY, and
GF_SURFACE_VYUY formats.

GFMXENCH264FETCHNAL

This structure is used by **GFMxEncH264FetchNAL()**.

GFMXENCH264FETCHIMAGE Structure

```
typedef struct GFMXENCH264FETCHNAL_TAG {
PVOID       pBuf;
NvU32       uiSizeofBuf;
NvU16       *pNALsLengthBuf;
NvU32       uiSizeofNALsLB;
NvU32       uiUTContentSize;
NvU32       uiTimeOut;
NvU32       uiNALsInfo;
NvU32       uiAverageQP;
NvU32       uiTime;
NvU32       uiFetchedSize;
NvU32       uiNALsLBSize;
} GFMXENCH264FETCHNAL;
```

GFMXENCH264FETCHNAL Field Definitions

These following fields are filled in by the application

pBuf

GFMxEncH264 fills this buffer with an H264-compliant bitstream.

uiSizeofBuf

Number of bytes

***pNALLengthBuf**

NAL unit length in bytes. If the encoder application wants know each NAL unit's size, this field should be set. If this field is NULL, the GFMxEncH264API does not output any NAL unit size information

uiSizeofNALLB

Number of NvU16. Size of pNALLengthBuf

uiUTContentSize

The size in DWORDs of the bitstream that has not been transferred out of the application's bitstream buffer. Valid only when the application enables rate control.

For a source image from the CPU, the application must set this field based on the ideal transfer rate because the application may flush the whole buffer before feeding and fetching another picture. If more than half of uiBufferSize in GFMXENCH264SETSEQUENCE has not been transferred out, the GFMXEncAPI starts reducing the bit rate. If less than half, the GFMXEncAPI starts increases the bit rate.

uiTimeOut

In milliseconds.

If 0, return immediately if there is no VOP ready.

If 1, wait until VOP is ready

The following fields are filled in by the `GFMxEncH264API`

uiImageInfo

See "GFMXENCH264FETCHNAL Field Definitions" on page 481.

uiAverageQP

Average QP for current picture

uiTime

In milliseconds, the encoded picture's time stamp. First picture may start from 0.

uiFetchedSize

Number of available bytes

uiNALLBSize

Number of available NvU16 in pNALLengthBuf

uiNALsInfo

`GF_MXENC_H264_NALS_PORTION_NAL`

`0x00000001`

The last NAL unit in pBuf only contains part of the NAL unit data. The last entry in pNALsLengthBuf is the size of this partial NAL unit size. The application must call to fetch the rest of data

uiNALsInfo
GF_MXENC_H264_NALS_MORE_NALS 0x00000002 More NALs are ready in the GFMxEncAPI internal buffer. The application should fetch as soon as possible.
GF_MXENC_H264_NALS_BEGIN_ACCESS_UNIT 0x00000004 The first NAL unit in pBuf is the first NAL unit of current access unit.
GF_MXENC_H264_NALS_END_ACCESS_UNIT 0x00000008 The last NAL unit in pBuf is the last NAL unit of current access unit. If any of GF_MXENC_H264_NALS_BEGIN_ACCESS_UNIT and GF_MXENC_H264_NALS_END_ACCESS_UNIT are set, those NAL units in pBuf are the middle parts of current access unit.

CHAPTER

15

H.264/AVC Decoder API (GFMxDecH264API)

GFMxDecH264 API Overview

The H.264/AVC decoder API handles H.264/AVC baseline-compliant bit streams for this version, but has been defined to handle all of the three profiles for future expansion.

The H.264/AVC decoder API has two sets of API—a high level API and a low level API.

Using the High Level API

The high level API has a built-in entropy decoder. It decodes the bit stream and performs error concealment if the bit stream has an error. This high level API calls the low level API internally. When using the high level API, the application can ignore the low level API.

Using the Low Level API

The low level API is essentially a macroblock decoder engine. It decodes macroblock by macroblock. When using the low level API, the application needs to handle the entropy decoding and error concealment. The low level API assumes that the information passed from caller is correct, and does not

have any error concealment functionality build in. The application which called the low level API must make sure that no macroblocks are missing for any picture.

GFMxDecH264 High Level API Reference

The H.264/AVC decoder high level API consists of the functions described under "GFMxDecH264API High Level Functions" on page 486, and the data structures described under "GFMxDecAPI Data Structures for High Level API" on page 491.

GFMxDecH264API High Level Functions

The GFMxDecH264API high level functions include the following:

- "GFMxDecH264DecSequence()" on page 486
- "GFMxDecH264DecPicture()" on page 487
- "GFMxDecH264DecResync()" on page 488
- "GFMxDecH264Set()" on page 490

GFMxDecH264DecSequence()

This API can be used to handle H.264/AVC bit streams.

GMMxDecH264DecSequence decodes the sequence header and returns the sequence characteristics.

Function Prototype

```
GF_RETTYPE GFMxDecH264DecSequence (
           GF_HANDLE               MXhandle,
           PGFMXDecH264DecSequence pSequence);
```

Parameters

`MXhandle` The handle from GFMxDecOpen()

`pSequence` Pointer to GFMXDecH264DECSEQUENCE

Return Values

GF_SUCCESS If successful

GF_ERROR If error

GFMxDecH264DecPicture()

This API can be used to handle H.264/AVC bit streams.

GFMxDecH264DecPicture decodes one picture from the bit stream. If GFMxDecH264DecPicture detects an error, it will perform error concealment. If the error is not concealable, GFMxDecH264DecPicture returns the error back to the application.

Function Prototype

```
GF_RETTYPE GFMxDecH264DecPicture (
           GF_HANDLE             MXhandle,
           PGFMXDecH264DecPicture pPicture);
```

Parameters

MXhandle The handle from GFMxDecOpen()

pPicture Pointer to GFMXDecH264DECPICTURE

Return Values

GF_SUCCESS If successful

GF_ERROR_264DECPICTURE_DETECTED_SEQUENCE:
There is a sequence header in the bit stream. The application must decide if this sequence is an error or this is intended to be in the bit stream.

GF_ERROR_264DECPICTURE_END_OF_FILE:
No more bit streams from the application.

GF_ERROR_264DECPICTURE_CORRUPTED_PICTURE:
GFMxDecH264DecPicture encounters an un-recoverable error in the bit stream. The application should call **GFMxDecH264DecResync** API to skip this picture.

GFMxDecH264DecResync()

This API can be used to handle the H.264/AVC bit stream.

GFMxDecH264DecResync re-synchronizes to a specific position of a bit stream.

Function Prototype

```
GF_RETTYPE GFMxDecH264DecResync (
           GF_HANDLE  MxHandle,
           NvU32      ResyncOption)
```

Parameters

`MxHandle`: Handle specific to the GFMxDec API

`ResyncOption`: Specifies the resync position. One of the following:

`MXDEC_264_RESYNC_FORWARD_NEAREST_SEQUENCE`:

Forward re-synchronize to the nearest sequence. If GFMxDec is right before a sequence, GFMxDecH264DecResync will return immediately. Otherwise, GFMxDecH264DecResync will move forward, parsing more bits, until it reaches the first bit of the next sequence. This option is useful for certain bit streams that have more than one sequence.

`MXDEC_264_RESYNC_FORWARD_NEAREST_PICTURE`:

Forward re-synchronize to the nearest picture. If GFMxDec is right before a picture, GFMxDecH264DecResync will return immediately. Otherwise, GFMxDecH264DecResync will move forward, parsing more bits, until it reaches the first bit of the next picture.

`MXDEC_264_RESYNC_FORWARD_NEAREST_IDR_PICTURE`:

Forward re-synchronize to the nearest I DR picture. If GFMxDec is right before an IDR picture, GFMxDecH264DecResync will return immediately. Otherwise, GFMxDecH264DecResync will move forward, parsing more bits, until it reaches the first bit of the next IDR picture.

`MXDEC_264_RESYNC_FORWARD_NEXT_SEQUENCE`:

Forward re-synchronize to the next sequence. If GFMxDec is right before or in the middle of sequence, GFMxDecH264DecResync will move forward, parsing more bits, until it reaches the first bit of the next sequence.

Parameters

MXDEC_264_RESYNC_FORWARD_NEXT_PICTURE:

Forward re-synchronize to the next picture. If GFMxDec is right before or in the middle of the picture, GFMxDecH264DecResync will move forward, parsing more bits, until it reaches the first bit of the next picture. The application can keep calling GFMxDecH264DecResync with this option to skip the picture.

MXDEC_264_RESYNC_FORWARD_NEXT_IDR_PICTURE:

Forward re-synchronize to the next IDR picture. If GFMxDec is right before or in the middle of picture, GFMxDecH264DecResync will move forward, parsing more bits, until reaches the first bit of next IDR picture. Application can keep calling GFMxDecH264DecResync with this option to skip non-IDR pictures. This is useful for fast forwarding, but it is not recommended to use this way if application can get the picture boundary and type from file header.

MXDEC_264_RESYNC_BACKWARD_NEAREST_SEQUENCE:

Backward re-synchronize to the nearest sequence. If GFMxDec is right before a sequence, GFMxDecH264DecResync will return immediately. Otherwise, GFMxDecH264DecResync will move backward, parsing more bits, until reaches the first bit of a sequence.

MXDEC_264_RESYNC_BACKWARD_NEAREST_PICTURE:

Backward re-synchronize to the nearest picture. If GFMxDec is right before a picture, GFMxDecH264DecResync will return immediately. Otherwise, GFMxDecH264DecResync will move backward, parsing more bits, until reaches the first bit of a picture.

MXDEC_264_RESYNC_BACKWARD_NEAREST_IDR_PICTURE:

Backward re-synchronize to the nearest IDR picture. If GFMxDec is right before an IDR picture, GFMxDecH264DecResync will return immediately. Otherwise, GFMxDecH264DecResync will move backward, parsing more bits, until reaches the first bit of an IDR picture.

MXDEC_264_RESYNC_BACKWARD_PREVIOUS_SEQUENCE:

Backward re-synchronize to the previous sequence. If GFMxDec is right before or in the middle of sequence, GFMxDecH264DecResync will move backward, parsing more bits, until reaches the first bit of previous sequence.

Parameters

MXDEC_264_RESYNC_BACKWARD_PREVIOUS_PICTURE:

Backward re-synchronize to the previous picture. If GFMxDec is right before or in the middle of picture, GFMxDecH264DecResync will move backward, parsing more bits, until it reaches the first bit of the picture.

MXDEC_264_RESYNC_FORWARD_NEXT_IDR_PICTURE:

Forward re-synchronize to the next IDR picture. If GFMxDec is right before or in the middle of a picture, GFMxDecH264DecResync will move backward, parsing more bits, until it reaches the first bit of an IDR picture. The application can keep calling GFMxDecH264DecResync with this option to skip non-IDR pictures. This is useful for fast rewinding, but it is not recommended to use this method if the application can get the picture boundary and type from file header.

Return Values

GF_SUCCESS: If success

264DECRESYNC_END_OF_FILE:

GFMxDecH264DecResync reached the end of the file (forward) or the beginning of the file (backward), but still cannot find the re-synchronized position.

GFMxDecH264Set()

This API can be used to handle H.264/AVC bit streams.

GFMxDecH264Set can be used to set up a callback function. GFMxDec calls that callback function to request a portion of the bit stream from the application. GFMxDecH264Set() can be used to pass a list of surfaces for H.264.

Function Prototype

```
GF_RETTYPE GFMxDecH264Set (
GF_HANDLE MxHandle,
NvU32     uiFeature,
void*     pInfo )
```

Parameters

MxHandle:	Handle specific to the GFMxDec API
uiFeature:	MXDEC_SET_READBITSTREAM, Set up the readbitstream callback function.
pInfo:	Pointer to PGFMXDECCALLBACK
uiFeature:	MXDEC_SET_SURFACE_LIST, Set up a list of surfaces for H.264 decoder
pInfo:	Pointer to an array of PGFRMSURFACE

Return Values

GF_SUCCESS:	If success
GF_ERROR:	If error

GFMxDecAPI Data Structures for High Level API

The GFMxDecH264API data structures for the high-level API include the following:

- "GFMXDecH264DECSEQUENCE" on page 491
- "GFMXDecH264DECPICTURE" on page 492

GFMXDecH264DECSEQUENCE

This structure holds H.264/AVC sequence information. This whole structure is filled by GFMxDecH264DecSequence.

GFMXDecH264DECSEQUENCE Structure

```
typedef struct _GFMXDecH264DECSEQUENCE {
Nvu32    SequenceInfo;
NvU16    profile_idc;
NvU16    level_idc;
NvU16    seq_parameter_set_id;
NvU16    num_ref_frames;
NvU8     constraint_set1_flag;
NvU8     constraint_set2_flag;
NvU16    pic_width_in_mbs_minus1;
NvU16    pic_height_in_map_units_minus1;
```

```
GFRECT   CroppingRect;
} GFMXDecH264DECSEQUENCE;
```

GFMXDecH264DECSEQUENCE Sequence Info Field Definition

SequenceInfo	
MXDEC_264_DEC_SEQUENCE_CONSTRAINT_SET0_FLAG If constraint_set0_flag is on in the bit stream, this flag is on.	0x00000001
MXDEC_264_DEC_SEQUENCE_CONSTRAINT_SET1_FLAG If constraint_set1_flag is on in the bit stream, this flag is on.	0x00000002
MXDEC_264_DEC_SEQUENCE_CONSTRAINT_SET2_FLAG If constraint_set2_flag is on in the bit stream, this flag is on.	0x00000004
MXDEC_264_DEC_SEQUENCE_FRAME_CROPPING_FLAG If this flag is set, constraint_set2_flag is on in the bit stream, this flag is on CroppingRect has the cropping rectangle area.	0x00000008

GFMXDecH264DECPICTURE

Structure to hold H.264/AVC picture information. This whole structure is filled by GFMxDecH264DecSequence.

GFMXDecH264DECPICTURE Structure

```
typedef struct _GFMXDecH264DECPICTURE {
//the following fields are set by the application:
NvU32        uiPictureOption;
PGFRECT      pPictureRect;
//the following fields are set by GFMxDec API:
NvU32        uiPictureInfo;
PGFRMSURFACE reconstructedSurf;
             // This is the surface to hold the decoded picture
NvU32        uiPictureOrderCount; //set by GFMxDec API
NvU32        uiTimeStamp; //in term of millisecond
NvU8         *pMBInfo;
             //each element tells the particular MB is wrong or
               not.
} GFMXDecH264DECPICTURE;
```

GFMXDecH264DECPICTURE Field Definitions

The following fields are set by the application.

uiPictureOption

MXDEC_264_DEC_PICTURE_SPECIFY_SURF_RECT	0x00000001

The current picture is IDR.

pPictureRect

Specifies a rectangle area for GFMxDec to output the decoded image.

This field takes effect only when **MXDEC_H264_DEC_PICTURE_SPECIFY_SURF_RECT** flag is set.

This rectangle area should be within the decoding surface. The rectangle top, left should be aligned by application. Application should call GFMxDecGetAttribute with **MXDEC_ATTR_264_DEC_PICTURE_RECT_TOP_ALIGNMENT** and **MXDEC_ATTR_264_DEC_PICTURE_RECT_LEFT_ALIGNMENT** attributions to get the required alignment.

This rectangle's width and height should exactly match the image width and height which has been coded in the bit stream.

If **pPictureRect** is NULL, GFMxDec will use the surface dimension as the decoded image dimension,

The following field is set by GFMxDecAPI

PictureInfo

MXDEC_264_DEC_PICTURE_IDR	0x00000001

The current picture is IDR.

MXDEC_264_DEC_PICTURE_BAD_MB	0x00000002

The current picture has at least one MB wrong

GFMxDecH264 Low Level API Reference

The H.264/AVC decoder low level API consists of the functions described under "GFMxDecH264 API Low Level Functions" on page 494, and the data structures described under "GFMxDecH264 Data Structures for Low Level API" on page 497.

GFMxDecH264 API Low Level Functions

The GFMxDecH264API low level functions include the following:

GFMxDecH264SetSequence()

The GMMxDecH264SetSequence sets up the current decoding bit stream sequence's characteristic.

Function Prototype

```
GF_RETTYPE GFMxDecH264SetSequence (
GF_HANDLE MXhandle,
PGFMXDecH264Sequence pSequence);
```

Parameters

`MXhandle`	The handle from GFMxDecOpen()
`pSequence`	Pointer to GFMXDecH264SEQUENCE

Return Values

`GF_SUCCESS`	If successful
`GF_ERROR`	If error

GFMxDecH264SetPicture()

The GMMxDecH264SetPicture sets up rgw current decoding picture's characteristic. After calling this API, the application must pass all of the slices in the picture to GFMXDEC.

Function Prototype

```
GF_RETTYPE GFMxDecH264SetPicture(
GF_HANDLE MXhandle,
PGFMXDecH264Picture pPicture);
```

Parameters

MXhandle	The handle from GFMxDecOpen()
pPicture	Pointer to GFMXDecH264PICTURE

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFMxDecH264SetSlice()

The GMMxDecH264SetSlice sets up the current decoding slice's characteristic. After calling this API, the application must pass all of the macroblock in this slice to GFMXDEC through GFMxDecH264SetMBs.

Function Prototype

```
GF_RETTYPE GFMxDecH264SetSlice(
GF_HANDLE MXhandle,
PGFMXDecH264SLICE pSlice);
```

Parameters

MXhandle	The handle from GFMxDecOpen()
pSlice	Pointer to GFMXDecH264SLICE

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFMxDecH264SetMBs()

The GMMxDecH264SetMBs performs inverse scan, inverse quantization, inverse transform, intra prediction, motion compensation and de-blocking to each macroblock.

Function Prototype

```
GF_RETTYPE GFMxDecH264SetMBs(
GF_HANDLE     MXhandle,
PGFMXDecH264MB pMBs,
NvU32         uiNumofMBs );
```

Parameters

MXhandle	The handle from GFMxDecOpen()
pMBs	Pointer to a buffer of GFMXDecH264MB
uiNumofMBs	Number of GFMXDecH264MB in the pMBs buffer. The minimum is 1, the maximum is the number of macroblocks in one slice.

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFMxDecH264 Data Structures for Low Level API

The GFMxDecH264API data structures for the low-level API include the following:

GFMXDECH264SEQUENCE

This structure holds H.264/AVC sequence information.

GFMXDECH264SEQUENCE Structure

```
typedef struct _GFMXDecH264SEQUENCE {
Nvu32        SequenceInfo;
NvU16        pic_width_in_mbs_minus1;
NvU16        pic_height_in_map_units_minus1;
} GFMXDecH264SEQUENCE;
```

GFMXDECH264SEQUENCE SequenceInfo Field Definition

SequenceInfo	
MXDEC_264_SEQUENCE_FRAME_MB_ONLY_FLAG If **frame_mb_only_flag** is on, this flag must be on.	0x00000001
MXDEC_264_SEQUENCE_MB_ADAPTIVE_FRAME_FIELD_FLAG If **mb_adaptive_frame_filed_flag** is on, this flag must be on.	0x00000002
MXDEC_264_SEQUENCE_DIRECT_8X8_INFERENCE_FLAG If **direct_8x8_inference_flag** is on, this flag must be on.	0x00000004

GFMXDECH264PICTURE

This structure holds H.264/AVC picture information.

GFMXDECH264PICTURE Structure

```
typedef struct _GFMXDecH264PICTURE {
Nvu32        PictureInfo;
PGFRMSURFACE reconstructedSurf;
             //surface to hold the decoded picture
PGFRECT      pRect;
Nvu8         weighted_bidpred_idc;
NvS8         chroma_qp_index_offset;
} GFMXDecH264PICTURE;
```

GFMXDecH264PICTURE PictureInfo Field Definition

PictureInfo	
GF_MXDEC_H264_PICTURE_WEIGHTED_PRED_FLAG If **weighted_pred_flag** is on, this flag must be on.	0x00000001
GF_MXDEC_H264_PICTURE_DEBLOCKING_FILTER_CONTROL_PRESENT_FLAG If **deblocing_filter_control_present_flag** is on, this flag must be on.	0x00000002
GF_MXDEC_H264_PICTURE_CONSTRAINED_INTRA_PRED_FLAG if **constrained_intra_pred_flag** is on, this flag must be on.	0x00000004
GF_MXDEC_H264_PICTURE_IDR	0x00000008
GF_MXDEC_H264_PICTURE_SPECIFY_SURFACE_RECT If this flag is set, pRect must be set. GFMxDec will reference the image in pRect area of reference surface and output the decoded image in the pRect area of reconstructedSurf. Otherwise, GFMxDec will use the entire surface area.	0x00000010

GFMXDECH264WEIGHT

This structure holds H.264/AVC weight information.

```
typedef struct _GFMXDecH264WEIGHT {
NvS8   luma_weight;
NvS8   luma_offset;
NvS8   Cb_weight;
NvS8   Cb_offset;
NvS8   Cr_weight;
NvS8   Cr_offset;
} GFMXDecH264WEIGHT;
```

GFMXDecH264SLICE

This structure holds H.264/AVC slice information.

GFMXDecH264SLICE Structure

```
typedef struct _GFMXDecH264SLICE {
NvU32              SliceInfo;
NvU16              first_mb_in_slice;
NvU8               slice_type;
NvU8               num_ref_idx_l0_active_minues1;
NvU8               num_ref_idx_l1_active_minues1;
NvU8               disable_deblocking_filter_idc ;
NvU8               slice_alpha_c0_offset_div2;
NvU8               slice_beta_offset_div2;
PGFRMSURFACE       ref_l0_surfaces[16];
PGFRMSURFACE       ref_l1_surfaces[16];
GFMXDecH264WEIGHT   weighttableL0[16];
GFMXDecH264WEIGHT   weighttableL1[16];
} GFMXDecH264SLICE;
```

GFMXDecH264SLICE SliceInfo Field Definition

SliceInfo	
MXDEC_264_SLICE_FILED_PIC_FLAG	0x01
When **filed_pic_flag** is on, this flag must be set.	
MXDEC_264_SLICE_BOTTOM_FILED_FLAG	0x02
When **bottom_field_flag** is on, this flag must be set.	
MXDEC_264_SLICE_DIRECT_SPATIAL_MV_PRED_FLAG	0x04
When **DIRECT_SPATIAL_MV_PRED_FLAG** is on, this flag must be set.	

GFMXDecH264I4X4LPREDMODE

This structure holds the H.264/AVC Intra_4x4 luma prediction mode.

```
typedef struct _GFMXDecH264I4X4LPREDMODE {
NvU32     uiPredMode0to7;
```

```
          //Luma4X4 Blk 0 to 7's prediction modes.
            Please use MAKE4X4PREDMODE macro to fill up this
             field.
NvU32      uiPredMode8to15;
            //Luma4X4 Blk 8 to 15's prediction modes.
             Please use MAKE4X4PREDMODE macro to fill up this
             field.
} GFMXDecH264I4X4LPREDMODE;
```

GFMXDecH264MV

This structure holds the H.264/AVC Motion Vector.

```
typedef struct _GFMXDecH264MV{
NvS16  Horizontal;  //Horizonal motion vector component.

NvS16  Vertical;   //Vertical motion vector component.
} GFMXDecH264MV;
```

GFMXDecH264COEF

This structure holds the H.264/AVC Coefficient.

```
typedef struct _GFMXDecH264COEF{
NvU8  uiLast; //This last non-zero coefficient in current block
NvU8  uiRUN;  //How many zeroes precede this coefficient
NvS16 iLEVEL; //This coefficient's level value
} GFMXDecH264COEF;
```

GFMXDecH264REFIDX

This structure holds the H.264/AVC reference index.

```
typedef struct _GFMXDecH264REFIDX{
NvU8  Ref_idx_l0[4]; //PartIdx's reference index
NvU8  Ref_idx_l1[4]; //PartIdx's reference index
} GFMXDecH264REFIDX;
```

GFMXDecH264SUBMBTYPE

This structure holds the H.264/AVC sub_mb_type.

```
typedef struct _GFMXDecH264SUBMBTYPE{
NvU8  sub_mb_type[4]; //PartIdx's sub_mb_type
} GFMXDecH264SUBMBTYPE;
```

GFMXDecH264MB

This structure holds H.264/AVC macroblock information.

GFMXDecH264MB Structure

```
typedef struct _GFMXDecH264MB {
NvU16  uiTotalSize;
       //Total size in bytes for the current macroblock,
         including this structure itself and the required
         structure following behind of this structure.
         Reference the data following GFMXDecH264MB.
NvU16  uiTotalCOEFs;  // total Coefs for this macroblock
NvU32  curMBAddress;  // current macroblock address
NvU8   MBInfo;        // this is the mb_type in bit stream and
                         mb_field_decoding_flag
NvU8   uiQP;          //this is QPy
NvU8   coded_block_pattern;
        //This field is valid only when current macroblock is
          not Intra_16x16
NvU8  Intra_chroma_pred_mode;
        //This field is valid only when current macroblock is
          Intra
GFMXDecH264SUBMBTYPE sub_mb_type;
union {
     GFMXDecH264I4X4LPREDMODE I4X4LPredMode; //Intra_4X4 only
     GFMXDecH264REFIDX        RefIdx;        //Inter only
     }
} GFMXDecH264MB;
```

GFMXDecH264MB Field Definitions

MBInfo

MXDEC_264_MB_FILED_DECODING_ENABLE 0x80

When mb_field_decoding_flag is on, this flag must be set.

The data following GFMXDecH264MB:

If the current macroblock is a skipped macroblock, no further data is needed.

If the current macroblock is an Inter macroblock, and is using picture list 0 only, the required number of GFMXDecH264MV are following.

If the current macroblock is using picture list 1 only, the required number of GFMXDecH264MV are following.

If the current macroblock is using both picture list 0 and list 1, the required number of GFMXDecH264MV for list 0, and the required number of GFMXDecH264MV for list 1 are following GFMXDecH264MB.

For both Intra and Inter macroblocks, the last portion of data which follows GFMXDecH264MB is GFMXDecH264MB.uiTotalCOEFs of GFMXDecH264COEF.

Attributes

`GF_MXDEC_ATTR_H264_PRE_DETERMINED_BOUNDARY 0x00000001`

The application will determine the decoding boundary.

For MPEG4, this boundary means VOL and VOP boundary.

For H.264/AVC, this boundary means sequence and picture boundary.

For certain file formats, the boundary information is stored in the video file. The application can easily use this information to determine the boundary.

If the application wants the API to detect the boundary, the application should not disable this attribute. This is mainly for the streaming case, as the application does not have the bounday information readily available. The API will automatically detects those boundaries, so the application does not need to parse the bitstream to detect those boundaries.

//*`pInfo`: 1: The application will determine the decoding boudary

//*`pInfo`: 0: The application will not determine the decoding boundary

`GF_MXDEC_H264_ATTR_DISPLAY_REORDERING 0x00000002`

Enforce display ordering for sequences containing out of order frames.

`GF_MXDEC_ATTR_H264_DEC_PICTURE_RECT_TOP_ALIGNMENT 0x00000007`

This attribute only can be used for GFMxDecGetAttribute. If the application wants to set pDestRect in GFMXDECPICTURE or GFMXDecH264DECPICTURE, the application must use this attribute to get alignment information and align the rectangle top and left corner accordingly.

//*`pInfo`: Top field alignment in term of lines

`GF_MXDEC_ATTR_H264_DEC_PICTURE_RECT_LEFT_ALIGNMENT 0x00000008`

This attribute only can be used for GFMxDecGetAttribute. If the application wants to set pDestRect in GFMXDECPICTURE or GFMXDecH264PICTURE, the application must use those attributes to get alignment information and align the rectangle top and left corner accordingly.

//*`pInfo`: Left field alignment in term of pixels

`GF_MXDEC_ATTR_H264_DEC_SAVE_STATE 0x00000009`

Save the decoder internal state information. This attribute only can be used for GFMxDecGetAttribute.

//*`pInfo`: State handle

GF_MXDEC_ATTR_H264_DEC_RESTORE_STATE	0x0000000a

Restore the decoder internal state information. This attribute only can be used for GFMxDecSetAttribute.

//*pInfo: state handle

GF_MXDEC_ATTR_H264_DEC_DELETE_STATE	0x0000000b

Delete the particular state handle. This attribute only can be used for GFMxDecSetAttribute.

//*pInfo: State handle

CHAPTER

16

I²C Bus API (GFI2CAPI)

Overview

The I²C (Inter-Integrated Circuit) specification is a standard serial bus protocol that describes two serial lines, SCL and SDA, that control the transfer of bytes of data. Each NVIDIA GoForce media processor has a dedicated hardware I²C master module that allows these byte transfers to be sent and received efficiently. Most of the cameras and other imaging devices that are connected to GoForce media processors are configured through the I²C bus. Usually, the camera's register settings, which control the resolution, brightness, and so on, are set up through the I²C bus.

GFI2CAPI Reference

The NVIDIA GFI2CAPI reference consists of the GFI2CAPI functions.

GFI2CAPI Functions

The GoForce I^2C bus API provides the five functions listed below:

GFI2CGetProperty()

This function returns the I^2C properties and capabilities.

Function Prototype

```
GF_RETTYPE  GFI2CGetProperty(
    GF_HANDLE    I2CHandle,
    PGFPROPERTY  pI2CProp );
```

Parameters

`I2CHandle`	Handle specific to the GFI2CAPI.
`pI2CProp`	Pointer to a "GFPROPERTY Structure" on page 10.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFI2CWrite()

This routine writes out **len** bytes of data during one I2C transfer.

Function Prototype

```
GF_RETTYPE  GFI2CWrite(
    GF_HANDLE  I2CHandle,
    NvU32      slaveAddress,
    NvU8       *pDataBytes,
    NvU32      len,
    NvU32      delayInMSec );
```

Parameters

I2CHandle	Handle specific to the GFI2CAPI.
slaveAddress	Target I2C slave address.
*pDataBytes	Pointer to the data bytes to be written out on the I2C transfer.
len	Number of bytes to be transferred.
delayInMSec	Delay in milliseconds between each byte during the I2C transfer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFI2CRead()

GFI2CRead() reads in **len** or fewer bytes of data in one I2C transfer.

Function Prototype

```
GF_RETTYPE  GFI2CRead(
    GF_HANDLE  I2CHandle,
    NvU32      slaveAddress,
    NvU8       *pDataBytes,
    NvU32      len,
    NvU32      delayInMSec );
```

Parameters

I2CHandle	Handle specific to the GFI2CAPI.
slaveAddress	Target I2C slave address.

Parameters (continued)

*pDataBytes	Pointer to stored data bytes read in during the I²C transfer.
len	Number of bytes to be transferred.
delayInMSec	Delay in milliseconds between each byte during the I²C transfer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFI2CRestartRead()

This function reads in len or fewer bytes of data from an I²C slave device. It differs from the GFI2CRead() function in that it does so with a combination of an I²C write and read. There is an I²C write of the **slaveAddress** and **index** followed by a START condition, and then an I²C read transaction, and finally a STOP condition. There is no STOP condition in-between the write and read transactions.

Function Prototype

```
GF_RETTYPE GFI2CRead
          ( GF_HANDLE I2CHandle,
            NvU32     slaveAddress,
            NvU32     index,
            NvU8      *pDataBytes,
            NvU32     len,
            NvU32     delayInMSec );
```

Parameters

I2CHandle;	GFI2CAPI specific handle
slaveAddress;	Target I2C slave address
Index;	Index register to be read
*pDataBytes	Pointer to store data bytes read from one I²C transfer
len;	Number of bytes to be transferred
delayInMSec;	Delay in milliseconds between each byte during I²C transfer

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFI2CScan()

GFI2CScan() attempts to write to **slaveAddress** on an I2C slave device and waits for an ACK response. By repeatedly calling this function, an application may be able to determine the slave address for the target device.

It is important to note that this procedure may not work on all slave devices because the routine generates a side effect when the sent **slaveAddress** is not recognized by the slave device. In this case, the function resets the I2C module so that next I2C transfer starts in a clean state.

Function Prototype

```
GF_RETTYPE  GFI2CScan(
    GF_HANDLE  I2CHandle,
    NvU16      slaveAddress );
```

Parameters

I2CHandle	Handle specific to the GFI2CAPI.
slaveAddress	Destination (*x* pixels from the left).

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFI2CPowerSwitch()

This routine requires source code customization by the hardware system developer. It is used to switch power on or off, and to reset a camera's (or any slave device's) configuration. Most cameras and slave devices require this function to be called before an I^2C bus transfer before they can respond to it.

Function Prototype

```
GF_RETTYPE  GFI2CPowerSwitch(
    GF_HANDLE  I2CHandle,
    NvU8       option );
```

Parameters

`I2CHandle`	Handle specific to the GFI2CAPI.
`option`	`0` Switch power off. `1` Switch power on.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFI2CSetAttribute()

This function sets certain attributes used by the GFI2CAPI. The aid parameter passes an enumeration GFI2CATTRIBUTES that selects the attribute operation. Each attribute has a set of defined values passed in the parameter **`attr`**, that it will operate on.

Function Prototype

```
GF_RETTYPE GFI2CSetAttribute
         ( GF_HANDLE I2CHandle,
           NvU32     aid,
           NvU32     attr);
```

Parameters

`I2CHandle;`	FI2CAPI specific handle
`aid;`	Attribute ID flag that indicates which attribute to change
`attr;`	Value passed to attribute is dependent on the operation required

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

GFI2CATTRIBUTES enumeration

GFI2C_ATTR_PULLUP	Sets the pull-up or pull down register for the SCL and SDA pins. Uses the following flags to enable or disable the pull-ups and pull downs:

```
define  SDA_PULLDOWN_DISABLE
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_D_PULLD_DISABLE

define  SDA_PULLDOWN_ENABLE
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_D_PULLD_ENABLE

define  SDA_PULLUP_DISABLE
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_D_PULLU_DISABLE

define  SDA_PULLUP_ENABLE
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_D_PULLU_ENABLE

define  SCL_PULLDOWN_DISABLE
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_CLK_PULLD_DISABLE

define  SCL_PULLDOWN_ENABLE
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_CLK_PULLD_ENABLE

define  SCL_PULLUP_DISABLE
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_CLK_PULLU_DISABLE

define  SCL_PULLUP_ENABLE
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_CLK_PULLU_DISABLE

define  CL_DEFAULT
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_CLK_PULLU_DEFAULT \
      | HOST1X_ASYNC_I2CPADCTRL_0_I2C_CLK_PULLD_DEFAULT

define  SDA_DEFAULT
        HOST1X_ASYNC_I2CPADCTRL_0_I2C_D_PULLU_DEFAULT \
      | HOST1X_ASYNC_I2CPADCTRL_0_I2C_D_PULLD_DEFAULT
```

GFI2CATTRIBUTES enumeration

GFI2C_ATTR_SW	Use software based GPIO manipulation for I2C transactions. No attr data required. Set attr = 0.
GFI2C_ATTR_HW	Use on-chip hardware for I2C transactions. No attr data required. Set attr = 0.

GFI2CSetClock()

This function sets the I^2C clock rate that is output to the I^2C slave device on the SCL pin. The parameter **selectclock** specifies the new I^2C clock in kilohertz. The default for **selectclock** is 100. If selectclock = 0, then the default will be set.

Function Prototype

```
GF_RETTYPE GF I2CSetClock
 ( GF_HANDLE  I2CHandle,
   NvU32      selectclock);
```

Parameters

I2CHandle;	GFI2CAPI specific handle
selectclock;	Select I2C clock in kHz

Return Values

GF_SUCCESS	If successful
GF_ERROR	If error

CHAPTER

17

Camera API (GFCameraAPI)

Overview

The GFCameraAPI provides a set of functions for manipulating a camera (imager) that is connected to a GoForce media processor. All data related to a particular camera is captured using this API. In addition, the information necessary to set up the GoForce media processor's video input port (VIP) through the GFVxAPI (see "Video API (GFVxAPI)" on page 219) is listed as part of camera information. Internally, the data intended for the camera is sent through an I^2C bus controlled by the GFI2CAPI. (See "I^2C Bus API (GFI2CAPI)" on page 507.)

For an example of using the GFCameraAPI, in conjunction with the GFI2CAPI and the GFVxAPI, see the VxPreview sample application.

GFCameraAPI Reference

The NVIDIA GFCameraAPI reference consists of the GFI2CAPI functions and data structures (see the "GFCameraAPI Data Structures" on page 519).

GFCameraAPI Functions

The GoForce Camera API provides the functions listed below:

GFCameraGetProperty()

The function returns the camera's properties and capabilities. These properties are returned in the **GFPROPERTY** structure.

Function Prototype

```
GF_RETTYPE  GFCameraGetProperty(
    GF_HANDLE    CamHandle,
    PGFPROPERTY  pProp );
```

Parameters

CamHandle	Handle specific to the GFCameraAPI.
pProp	Pointer to GFPROPERTY structure. See "GFPROPERTY" on page 10.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFCameraSetup()

This routine programs the camera's registers based on the data contained in the camera table.

Function Prototype

```
GF_RETTYPE  GFCameraSetup(
    GF_HANDLE           CamHandle,
    PGFCAMERATABLETYPE  pCamera,
```

Function Prototype (continued)

```
    NvU16                   resX,
    NvU16                   resY );
```

Parameters

CamHandle	Handle specific to the GFCameraAPI.
pCamera	Pointer to camera entry. See "GFCAMERATABLETYPE" on page 522.
resX	Horizontal resolution.
resY	Vertical resolution.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFCameraTableAlloc()

GFCameraTableAlloc() allocates the run-time camera table structure by parsing the camera information byte stream.

Function Prototype

```
GF_RETTYPE  GFCameraTableAlloc(
    GF_HANDLE           CamHandle,
    NvU8                *pTable,
    PGFCAMERATABLETYPE  *ppCamera );
```

Parameters

CamHandle	Handle specific to the GFCameraAPI.
*pTable	Pointer to camera table.
*ppCamera	Pointer to camera entry. See "GFCAMERATABLETYPE" on page 522.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFCameraTableFree()

GFCameraTableFree() frees the camera table structure created by **GFCameraTableAlloc()**.

Function Prototype

```
GF_RETTYPE  GFCameraTableFree(
GF_HANDLE           CamHandle,
PGFCAMERATABLETYPE  *ppCamera );
```

Parameters

CamHandle	Handle specific to the GFCameraAPI.
*ppCamera	Pointer to camera entry. See "GFCAMERATABLETYPE" on page 522

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFCameraTableDump()

This function dumps the contents of the camera tables as ASCII data.

Function Prototype

```
GF_RETTYPE  GFCameraTableDump(
GF_HANDLE           CamHandle,
PGFCAMERATABLETYPE  pCamera );
```

Parameters

CamHandle	Handle specific to the GFCameraAPI.
pCamera	Pointer to camera entry. See "GFCAMERATABLETYPE" on page 522

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFCameraFind()

`GFCameraFind()` finds the right run-time camera table structure.

Function Prototype

```
GF_RETTYPE  GFCameraFind(
    GF_HANDLE           CamHandle,
    PGFCAMERATABLETYPE  pCamera,
    PGFCAMERATABLETYPE  *pWanted,
    NvU32               id,
    Char                *name,
    NvU16               resX,
    NvU16               resY );
```

Parameters

`CamHandle`	Handle specific to the GFCameraAPI.
`pCamera`	Pointer to the camera entry. See "GFCAMERATABLETYPE" on page 522.
`*pWanted`	Pointer to the found `GFCAMERATABLETYPE` structure.
`id`	Unique I^2C slave ID for the camera.
`*name`	Pointer to name.
`resX`	Horizontal resolution.
`resY`	Vertical resolution.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFCameraAPI Data Structures

These data structures are part of the GoForce Camera API:

GFCAMERA Definitions

GFCAMERA Definitions

```
/*
    GFCAMERA Instruction (Special)
*/
#define GFCAMERA_NONE                                   0x00
#define GFCAMERA_NORMAL_WRITE                           0x01
#define GFCAMERA_DELAY_MSEC                             0x02
#define GFCAMERA_NEED_MCLOCK                            0x03
```

GFCAMERAINSTRTYPE

This structure is used in the "GFCAMERARESOLUTIONTYPE" on page 521.

GFCAMERAINSTRTYPE Structure

```
typedef struct _GFCAMERAINSTRTYPE {
    NvU16  skip;
        // IMPORTANT: Multiple instructions must be in
        // ascending skip order.
    NvU8   type;
    NvU16  size;
    NvU32  data;
} GFCAMERAINSTRTYPE, *PGFCAMERAINSTRTYPE;
```

GFCAMERABAYERINFO

GFCAMERABAYERINFO Structure

```
typedef struct _GFCAMERABAYERINFO
{
    NvU16       ScanWidth;
    NvU16       ScanHeight;

    NvU16       ActiveLineStart;
    NvU16       ActiveLineWidth;
    NvU16       ActiveFrameStart;
    NvU16       ActiveFrameHeight;
    NvU8        hSyncEdge;
    NvU8        vSyncEdge;
    NvU16       bayerSel;
} GFCAMERABAYERINFO, *PGFCAMERABAYERINFO;
```

GFCAMERARESOLUTIONTYPE

This structure is used in the "GFCAMERATABLETYPE" on page 522.

GFCAMERARESOLUTIONTYPE Structure

```
typedef struct _GFCAMERARESOLUTIONTYPE {
    NvU16              x;
    NvU16              y;
    NvU8               numInstr;
    GFCAMERAINSTRTYPE  *pInstr;
} GFCAMERARESOLUTIONTYPE, *PGFCAMERARESOLUTIONTYPE;
```

GFCAMERATABLETYPE

This structure is used by many of the functions in the GFCameraAPI.

GFCAMERATABLETYPE Structure

```
#define GFCAMERA_NAME_SIZE                              32
typedef struct _GFCAMERATABLETYPE {
    NvU16  id;
    NvU16  ver;
    char   name[GFCAMERA_NAME_SIZE];
        // Variable name char string always ends with a 0.
    NvU32  VIPFlag;
        // See GFVxVIPSetVIP API.
    NvU32  VIPNewTiming;
        // See GFVxVIPSetVIP API.
    NvU32  VIPColorFormat;
        // See GFVxVIPSetVIP API.
    NvU8   VIPHOffset;
        // See GFVxVIPSetVIP API.
    NvU8   VIPVOffset;
        // See GFVxVIPSetVIP API.
    NvU8   numI2CBytes;
        // Bytes per transfer/packet.
    NvU8   numRes;
    GFCAMERARESOLUTIONTYPE     *pResData;
    NvU16  initDataSize;
    NvU8   *pInitData;
    struct _GFCAMERATABLETYPE  *pNext;
} GFCAMERATABLETYPE, *PGFCAMERATABLETYPE;
```

CHAPTER

18

File Device API (GFFDevAPI)

Overview

The architecture of the GFFDevAPI, the GoForce file device API, is depicted in Figure 9. The currently supported devices are as follows:

- **Standard C FILE I/O.** No separate file system is required because the API is mapped to standard C FILE I/O library functions.
- **Secure Digital (SD) device.** This file device type (usually a memory card) uses a simplified FAT file system implemented in the GFSDK.
- **Memory Disk (MD) or RAM Disk.** This file device type uses system memory to emulate a FAT file system.

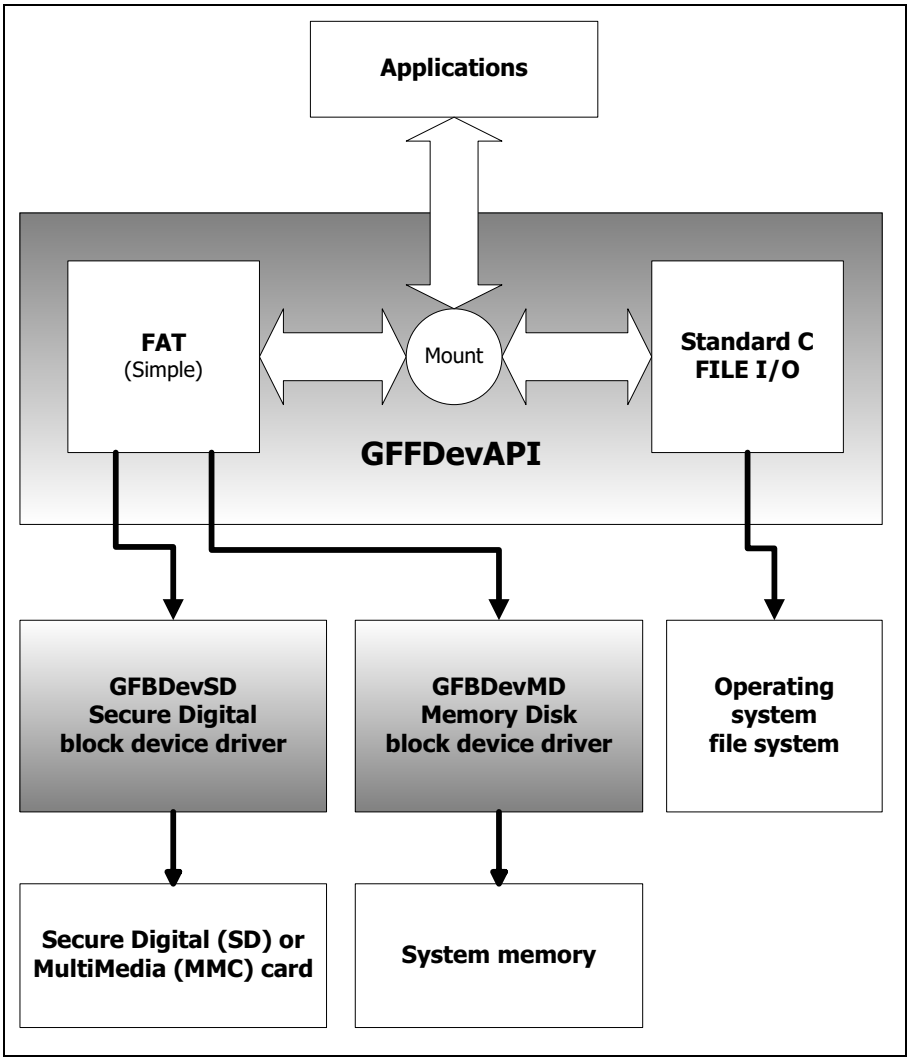


Figure 9. GFFDevAPI Architecture

GFFDevAPI Reference

The file device API consists of the GFFDevAPI functions, which are detailed in the next section.

GFFDevAPI Functions

The GFFDevAPI functions are as follows:

GFFDevMount()

This function is required for mounting and selecting the proper file device. The currently supported devices are the following:

- **Standard C FILE I/O.**
- **Secure Digital (SD) device.**
- **Memory Disk (MD) or Ram Disk.** For these devices, the **`RamDiskSize`** parameter has to be specified.

Function Prototype

```
GF_RETTYPE  GFFDevMount(
    GF_HANDLE  FDevHandle,
    NvU32      capType,
    NvU16      DeviceID,
    NvU16      DeviceRev,
    NvU32      RamDiskSize );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
capType	Capability type.
DeviceID	Device ID.
DeviceRev	Device revision.
RamDiskSize	Size of the memory or RAM block device in bytes. If capType is not GFFDEV_CAP_DEVICE_MD, this parameter is ignored.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevMount() Definitions

```
/*
    capType
*/
#define GFFDEV_CAP_STANDARD                    0x00010000
    // Standard C Library
#define GFFDEV_CAP_DEVICE_SD                   0x00000001
    // SD Device
#define GFFDEV_CAP_DEVICE_MD                   0x00000002
    // Memory Disk Device
#define GFFDEV_CAP_FAT                         0x00020000
    // FAT
```

GFFDevUnmount()

This function is used to unmount the proper file device.

Function Prototype

```
GF_RETTYPE  GFFDevUnmount(
    GF_HANDLE  FDevHandle );
```

Parameters

FDevHandle	Handle specific to the GFFDev.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevOpenFile()

This function opens the file specified by **fileName**. The file can be an existing file or a file newly created through this API. The returned file handle, which is associated with the opened file, can be used for reading and writing later.

If the mounted device is an SD device or an MD device, **fileSpec** can only be **"r"** or **"w"**.

If the mounted device is the standard C library, **fileSpec** can be **"r"**, **"w"**, or **"r+w"**.

Function Prototype

```
GF_RETTYPE  GFFDevOpenFile(
    GF_HANDLE  FDevHandle,
    Char       *fileName,
    Char       *fileSpec,
    GF_HANDLE  *pFileHandle );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
*filename	Pointer to a null-terminated string naming the file to be opened
*fileSpec	Specifies the action to be taken: • "r" open for read only • "w" open for write only • "r+w" open for read and write
*pFileHandle	Pointer to the returned file handle.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevCloseFile()

This function closes a file.

Function Prototype

```
GF_RETTYPE  GFFDevCloseFile(
    GF_HANDLE  FDevHandle,
    GF_HANDLE  *pFileHandle );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
*pFileHandle	Pointer to file handle.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevReadFile()

This function reads the number of bytes from a file. The returned count of bytes read is stored in **pCount** and the data is stored in the buffer pointed to by **pLoadBuf**.

Function Prototype

```
GF_RETTYPE  GFFDevReadFile(
    GF_HANDLE  FDevHandle,
    GF_HANDLE  filehandle,
    void       *pLoadBuf,
    NvU32      *pCount );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
filehandle	Pointer to file handle obtained from GFFDevOpenFile().
*pLoadBuf	Pointer to data buffer.
*pCount	Pointer to read count.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevWriteFile()

This function writes the specified number of bytes to a file. The returned count of bytes written is stored in **pCount**.

Function Prototype

```
GF_RETTYPE  GFFDevWriteFile(
    GF_HANDLE  FDevHandle,
```

Function Prototype

```
    GF_HANDLE  filehandle,
    void       *pLoadBuf,
    NvU32      *pCount );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
filehandle	Pointer to file handle.
*pLoadBuf	Pointer to data buffer.
*pCount	Pointer to write count.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevSeekFile()

This function moves the file pointer to a specified location within the file.

Function Prototype

```
GF_RETTYPE  GFFDevSeekFile(
    GF_HANDLE       FDevHandle,
    GF_HANDLE       FileHandle,
    NvU32           offset,
    GFFDEVSEEKTYPE  where );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
FileHandle	Pointer to file handle.
offset	Specifies the offset location if where is set to GFFDEV_SEEK_CUR.
where	Specifies the new file location: • GFFDEV_SEEK_SET: Beginning of file • GFFDEV_SEEK_CUR: Offset location • GFFDEV_SEEK_END: End of file

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevSeek Type Enumeration Type

```
/*
    Public GFFDevSeekFile Parameter enum
*/
typedef enum {
    GFFDEV_SEEK_SET = 0,
        // Set FileHandle to beginning of file.
    GFFDEV_SEEK_CUR,
        // Set FileHandle to specified offset location.
    GFFDEV_SEEK_END
        // Set FileHandle to end of file.
} GFFDEVSEEKTYPE;
```

GFFDevTellFile()

This function returns the current position of a file pointer.

Function Prototype

```
GF_RETTYPE  GFFDevTellFile(
    GF_HANDLE  FDevHandle,
    GF_HANDLE  FileHandle,
    NvU32      *pOffset );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
FileHandle	Pointer to file handle.
*pOffset	Pointer to store the returned file pointer.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevGetFileSize()

This function returns the file size.

Function Prototype

```
GF_RETTYPE  GFFDevGetFileSize(
    GF_HANDLE  FDevHandle,
    char       *fileName,
    NvU32      *pSize );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
*fileName	Pointer to a file name string.
*pSize	Pointer to a NvU32 to hold file size.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevRenameFile()

This function changes a file name from the contents of **oldFileName** to the contents of **newFileName**.

Function Prototype

```
GF_RETTYPE  GFFDevRenameFile(
    GF_HANDLE  FDevHandle,
    char       *oldFileName
    char       *newFileName );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
*oldFileName	Pointer to an old file name string.
*newFileName	Pointer to a new file name string.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevDeleteFile()

This function deletes an existing file.

Function Prototype

```
GF_RETTYPE   GFFDevDeleteFile(
  GF_HANDLE  FDevHandle,
  char       *fileName );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
*fileName	Pointer to a file name string.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDevFindFirstFile()

This function provides information about the first instance of a file that matches the name specified in the **fileSpec** argument.

Function Prototype

```
GF_RETTYPE  GFFDevFindFirstFile(
    GF_HANDLE       FDevHandle,
    char            *fileSpec,
    GFFDEVFILEINFO  *pFileInfo,
    GF_HANDLE       *FileHandle );
```

Parameters

FDevHandle	Handle specific to the GFFDev.
*fileSpec	Pointer to a file name string that can contain wild card characters (*).
*pFileInfo	Pointer to a GFFDEVFILEINFO structure that holds the file information after the call returns.
*FileHandle	Pointer to a file search handle that is used with GFFDevFindNextFile() and GFFDevCloseFile().

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFFDEVFILEINFO Structure

```
typedef struct _GFFDEVFILEINFO {
    unsigned  attrib;
    NvU32     time_create;    // -1 for FAT file systems
    NvU32     time_access;    // -1 for FAT file systems
    NvU32     time_write;
```

GFFDEVFILEINFO Structure (continued)

```
    NvU32     size;
    char      name[260];
} GFFDEVFILEINFO, *PGFFDEVFILEINFO;
```

GFFDevFindNextFile()

This function provides the information about the next instance of a file that matches the file named in the **`fileSpec`** argument in a previous call to **`GFFDevFindFirstFile()`**.

Function Prototype

```
GF_RETTYPE  GFFDevFindNextFile(
    GF_HANDLE       FDevHandle,
    GFFDEVFILEINFO  *pFileInfo,
    GF_HANDLE       *FileHandle );
```

Parameters

`FDevHandle`	Handle specific to the GFFDev.
`*pFileInfo`	Pointer to a `GFFDEVFILEINFO` structure that holds the file information after the call returns.
`*FileHandle`	Pointer to a file search handle that came from `GFFDevFindFirstFile()`.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFFDevFindCloseFile()

This function closes the specified search handle and releases the associated resources.

Function Prototype

```
GF_RETTYPE  GFFDevCloseFile(
    GF_HANDLE  FDevHandle,
    GF_HANDLE  *FileHandle );
```

Parameters

`FDevHandle`	Handle specific to the GFFDev.
`*FileHandle`	Pointer to a file search handle that came from `GFFDevFindFirstFile()`.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

CHAPTER

19

Image Signal Processing API (GFISPAPI)

Overview

As image resolution increases (three megapixels and up), image signal processing inside the imager module becomes complex and costly, and the required bandwidth to transfer processed data to the GPU increases.

One solution to this problem is to incorporate image signal processing (ISP[1]) capabilities inside the GPU, and let the imager send raw Bayer Pattern Data from the CMOS or CCD sensor to the GPU. This reduces bandwidth as ISP programming is easier and much faster compared to having the ISP unit inside the imager.

The ISP pipeline consists of various stages of image processing operations to enhance the image so that its digital representation is as close to the natural look as possible. The ISP pipeline consists of the following stages:

- Optical Black Compensation (OB)
- De-Knee Linearization
- Lens Shading Compensation

1. Image Signal Processing (ISP) is required to reduce imperfections caused by the process of converting light into digital samples of luminance and chrominance (YUV).

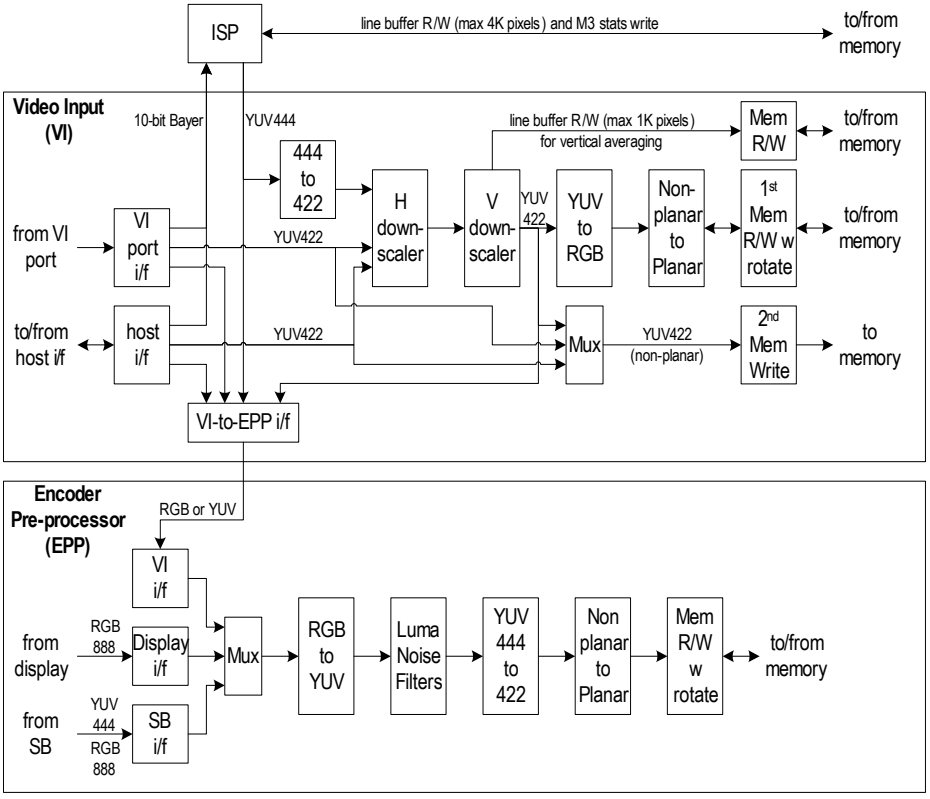
- White Balance Gain Adjustment (M1, M2, M3 statistics gathering)
- Bad Pixel Compensation
- Noise Reduction
- Demosaicing
- Edge Enhancement (M4 statistics gathering)
- Color Noise Reduction
- Color Correction
- Gamma Correction
- RGB to YUV conversion

Besides these pipeline stages, the ISP module also provides a statistics collection mechanism that can be applied to multiple regions of the image. Image statistics information can be used to compute Auto Exposure (AE), Auto White Balance (AWB) and Auto Focus (AF).

The GFISPAPI sets up parameters for each pipeline functional block. If needed, the user can also choose to bypass each individual functional block. Once all blocks are set up, image data flows through them automatically.

Figure 10, “Block Diagram of the ISP Interface,” on page 537 shows the ISP with respect to other input blocks such as video input (VI) and the encoder preprocessor (EPP).

Figure 11, “Internal ISP Data Flow,” on page 538 shows the data flow inside the ISP.



Notes:

1. First VI to memory interface can write the following data format which is typically used for video preview or thumbnail generation :
   a. YUV422 non-planar (16-bpp) and YUV422 planar with optional rotation
   b. YUV420 planar with optional rotation
   c. RGB888 (32-bpp) with optional rotation
   d. RGB565 (16-bpp) with optional rotation
   e. RAW (8-bpp) variable line length w /o rotation
2. Second VI to memory interface can write the following data format which is typically used as input to StretchBLT :
   a. YUV422 non-planar (16-bpp) and YUV422 planar with optional rotation
3. VI to EPP interface can transfer the following data format :
   a. YUV444 non-planar or RGB888
   b. YUV422 sent as YUV444 with UV duplication
   c. RGB565
   d. Raw 8-bit and 12-bit data
4. EPP to memory interface can write the following data format :
   a. YUV444 non-planar stored as 1-pixel/32-bit and YUV444 planar with optional rotation
   b. YUV422 non-planar stored as 2-pixel/32-bit and YUV422 planar with optional rotation
   c. YUV420 planar with optional rotation
   d. RGB888 (32-bpp) with optional rotation

Figure 10. Block Diagram of the ISP Interface

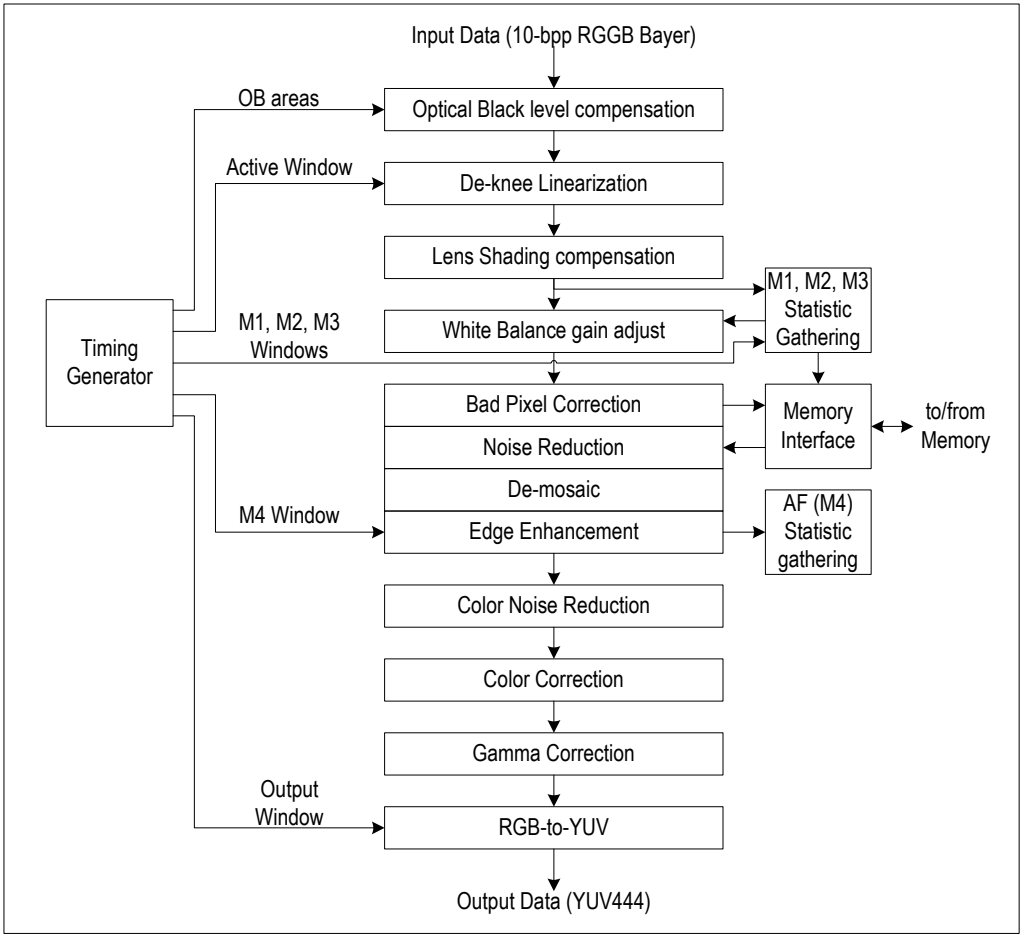


Figure 11. Internal ISP Data Flow

GFISPAPI Reference

The GFISPAPI consists of the functions described under "GFISPAPI Functions" on page 539 and the data structures described under "GFISPAPI Parameter Data Structures" on page 549.

GFISPAPI Functions

This section describes the following GFISPAPI functions:

GFISPGetProperty()

This function returns information about the GFISPAPI, including the following:

- GFISPAPI module version
- ISP capability
- ISP limit parameters

It is good practice to use this function to query for the API version and capabilities before using other GFISPAPI functions.

Function Prototype

```
GF_RETTYPE GFISPGetProperty(
    GF_HANDLE       IspHandle,
    PGFISPPROPERTY  pIspProp );
```

Parameters

`IspHandle`	Handle specific to the GFISPAPI .
`pIspProp`	Pointer to "GFISPPROPERTY Structure".

Returns

GF_SUCCESS	If successful
GF_ERROR	If error

GFISPGetProperty Definitions

```
#define GFISP_CAP_FORMAT_BAYER                0x00000001
#define GFISP_CAP_FORMAT_STACKED              0x00000002
#define GFISP_CAP_THREE_OB_REGIONS            0x00000004
#define GFISP_CAP_FOUR_OB_REGIONS             0x00000008
#define GFISP_CAP_FRAME_OB                    0x00000010
#define GFISP_CAP_LINE_OB                     0x00000020
#define GFISP_CAP_COLUMN_OB                   0x00000040
#define GFISP_CAP_DE_KNEE                     0x00000100
#define GFISP_CAP_LENSSHADING                 0x00000200
#define GFISP_CAP_WB                          0x00000400
#define GFISP_CAP_DEMOSIAC                    0x00000800
#define GFISP_CAP_EDGEENHANCE                 0x00001000
#define GFISP_CAP_BADPIXEL                    0x00002000
#define GFISP_CAP_NOISEREDUCTION1             0x00004000
#define GFISP_CAP_NOISEREDUCTION2             0x00008000
#define GFISP_CAP_COLORCORRECTION             0x00100000
#define GFISP_CAP_GAMMACORRECTION             0x00200000
#define GFISP_CAP_YUVCONVERSION               0x00400000
#define GFISP_CAP_M1                          0x01000000
#define GFISP_CAP_M2                          0x02000000
#define GFISP_CAP_M3                          0x04000000
#define GFISP_CAP_M4                          0x08000000
#define GFISP_CAP_NEGATIVEEFFECT              0x80000000
```

GFISPPROPERTY Structure

```
typedef struct  _GFISPPROPERTY{
    NvU32  size;
    NvU32  Version;
    NvU32  Cap;
    NvU32  MaxWidth;
    NvU32  MaxHeight;
} GFISPPROPERTY, *PGFISPPROPERTY;
```

GFISPPROPERTY Structure Values

`size`	Actual size of `GFISPPROPERTY` data structure. For the sake of future expansion, the caller should put the size of the structure into this field; GFISPAPI will not fill the structure beyond this size.
`Version`	Version of the GFISPAPI.
`Cap`	Combination of `GFISP_CAP_xxx` bits. Each bit corresponds to one ISP capability.
`MaxWidth`	Maximum image width.
`MaxHeight`	Maximum image height.

GFISPGetAttribute()

This function gets an attribute for a given ISP functional part.

To get parameters for specific ISP functions, see "GFISPGetParameter()" on page 544.

Function Prototype

```
GF_RETTYPE GFISPGetAttribute(
    GF_HANDLE        IspHandle,
    GFISPATTRIBUTES  aid,
    NvU32            *attr );
```

Parameters

`IspHandle`	Handle specific to the GFISPAPI.
`aid`	Attribute ID. One of "GFISPATTRIBUTES Enumeration Type" value, which corresponds to certain ISP functionality.
`attr`	32-bit attribute value.

Returns

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFISPSetAttribute()

This function sets an attribute for a given ISP functional part.

Under the NVIDIA ISP implementation, certain ISP functions can be turned on or off independent of parameter settings. In most cases, this function simply switches certain ISP functions on or off.

To set parameters for specific ISP functions, see "GFISPSetParameter()" on page 545.

Function Prototype

```
GF_RETTYPE GFISPSetAttribute(
    GF_HANDLE         IspHandle,
    GFISPATTRIBUTES   aid,
    NvU32             attr );
```

Parameters

IspHandle	Handle specific to the GFISPAPI.
aid	Attribute ID. One of "GFISPATTRIBUTES Enumeration Type" value, which corresponds to certain ISP functionality.
attr	32-bit attribute value.

Returns

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFISPATTRIBUTES Enumeration Type

```
typedef enum _GFISPATTRIBUTES {
    GFISP_ATTR_ENABLE_ISP,
        // The master switch of ISP.
        // 1 = Enable ISP processing.       0 = Disable.
    GFISP_ATTR_ENABLE_OB,
        // 1 = Enable OB.                   0 = Disable.
    GFISP_ATTR_ENABLE_FOB,
        // 1 = Enable frame based OB.       0 = Disable.
    GFISP_ATTR_ENABLE_FOB_SELFRESET,
        // 1 = Enable (reset every frame).  0 = Disable.
    GFISP_ATTR_ENABLE_FOB_TEMPORAL,
        // 1 = Enable temporal filtering.   0 = Disable.
    GFISP_ATTR_ENABLE_LOB,
        // 1 = Enable line based OB.        0 = Disable.
```

GFISPATTRIBUTES Enumeration Type

```
    GFISP_ATTR_ENABLE_DEKNEE,
        // 1 = Enable DeKnee (input linearization).
        // 0 = Disable.
    GFISP_ATTR_ENABLE_LENS_SHADING,
        // 1 = Enable lens shading correction.
        // 0 = Disable.
    GFISP_ATTR_ENABLE_WB,
        // 1 = Enable white balance.        0 = Disable.
    GFISP_ATTR_ENABLE_AWB,
        // 1 = Automatic/dynamic (real-time) white balance.
        // 0 = Preprogrammed mode white balance.
    GFISP_ATTR_AWB_HOLD,
        // Only effective on GFISP_ATTR_ENABLE_AWB=1.
        // 1 = Start applying current AWB gain value to all
        //     future frames.
        // 0 = Restore normal AWB mode.
    GFISP_ATTR_ENABLE_DEMOSAIC,
        // 1 = Enable demosaic processing.  0 = Disable.
    GFISP_ATTR_ENABLE_EDGE_ENHANCE,
        // 1 = Enable edge enhancement.     0 = Disable.
    GFISP_ATTR_ENABLE_NOISE_CONTROL1,
        // 1 = Enable noise control method 1 (built into the
        //     Demosaic unit).
        // 0 = Disable.
    GFISP_ATTR_ENABLE_BAD_PIXEL,
        // 1 = Enable bad pixel detection and concealment.
        // 0 = Disable.
    GFISP_ATTR_ENABLE_COLOR_CORRECTION,
        // 1 = Enable color correction.     0 = Disable.
    GFISP_ATTR_ENABLE_NOISE_CONTROL2,
        // 1 = Enable noise control method 2 (built into the
        //     color correction unit).
        // 0 = Disable.
    GFISP_ATTR_ENABLE_GAMMA_CORRECTION,
        // 1 = Enable gamma correction.     0 = Disable.
    GFISP_ATTR_ENABLE_YUV_CONVERSION,
        // 1 = Enable RGB-YUV conversion.   0 = Disable.
    GFISP_ATTR_ENABLE_NEGATIVE_EFFECT,
        // 1 = Negate whole output image.
        // 0 = Normal image.
```

GFISPATTRIBUTES Enumeration Type

```
    GFISP_ATTR_ENABLE_M2,
        // 1 = Enable M2 statistics gathering.
        // 0 = Disable.
    GFISP_ATTR_ENABLE_M3,
        // 1 = Enable M3 statistics gathering.
        // 0 = Disable.
    GFISP_ATTR_ENABLE_M4,
        // 1 = Enable M4 statistics gathering.
        // 0 = Disable.
    GFISP_ATTR_ENABLE_STATS_RAISE,
        // 1 = Generate raise vector when statistics gathering
        //     is complete.
        // 0 = Disable.
    GFISP_ATTR_ENABLE_STATS_INT,
        // 1 = Generate interrupt when statistics gathering
        //     is complete.
        // 0 = Disable.
    GFISP_ATTR_STATS_LINE_END,
        // The line at which statistics gathering is complete.
        // When the line counter reaches STATS_LINE_END, the
        // raise vector and/or interrupt are generated
        // (if enabled).
} GFISPATTRIBUTES;
```

GFISPGetParameter()

This function gets parameters for a given ISP functional part.

Also refer to "GFISPGetAttribute()" on page 541.

Function Prototype

```
GF_RETTYPE GFISPGetParameter(
    GF_HANDLE        IspHandle,
    GFISPPARAMETERS  pid,
    unsigned         *pSize,
    void             *pPara );
```

Parameters

IspHandle	Handle specific to the GFISPAPI.
pid	Parameter ID. A "GFISPPARAMETERS Enumeration Type" value, which corresponds to certain ISP functionality.
pSize	Pointer to the actual GFISPAPI data structure size, depending on pid. Can be NULL on GetParameter().
pPara	Pointer to certain GFISPAPI data structure, depending on pid. Can be NULL on GetParameter().

Returns

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFISPSetParameter()

This function sets parameters for a given ISP functional part.

Parameters can be set independent of the switch on or off state.

Also refer to "GFISPSetAttribute()" on page 541.

Function Prototype

```
GF_RETTYPE GFISPSetParameter(
    GF_HANDLE        IspHandle,
    GFISPPARAMETERS  pid,
    unsigned         size,
    void             *pPara );
```

Parameters

IspHandle	Handle specific to the GFISPAPI.
pid	Parameter ID. A "GFISPPARAMETERS Enumeration Type" value, which corresponds to certain ISP functionality.
size	Actual GFISPAPI data structure size, depending on pid.
pPara	Pointer to certain GFISPAPI data structure, depending on pid. Can be NULL on GetParameter().

Returns

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFISPPARAMETERS Enumeration Type

```
typedef enum _GFISPPARAMETERS {
    GFISP_PARA_TIMING,       // pPara = GFISPIMGTIMING
    GFISP_PARA_M1WINDOW,     // pPara = GFISPM1WINDOW
    GFISP_PARA_M2WINDOW,     // pPara = GFISPM2WINDOW
    GFISP_PARA_M3WINDOW,     // pPara = GFISPM3WINDOW
    GFISP_PARA_M4WINDOW,     // pPara = GFISPM4WINDOW
    GFISP_PARA_OB,           // pPara = GFISPOBCONTROL
    GFISP_PARA_DEKNEE,       // pPara = GFISPDEKNEECONTROL
    GFISP_PARA_LENSSHADING,  // pPara = GFISPLENSSHADING
    GFISP_PARA_WB,           // pPara = GFISPWBCONTROL
    GFISP_PARA_DEMOSAIC,     // pPara = GFISPDEMOSAICCONTROL
    GFISP_PARA_EDGEENHANCE,  // pPara = GFISPEDGEENHANCECONTROL
    GFISP_PARA_NOISECONTROL1,
        // pPara = GFISPNOISECONTROL1
    GFISP_PARA_NOISECONTROL2,
        // pPara = GFISPNOISECONTROL2
    GFISP_PARA_BADPIXEL,     // pPara = GFISPBADPIXELCONTROL
    GFISP_PARA_COLORCORRECT, // pPara = GFISPCOLORCORRECTION
    GFISP_PARA_GAMMACORRECT, // pPara = GFISPGAMMACORRECTION
    GFISP_PARA_YUVCONVERT    // pPara = GFISPYUVCONVERSION
} GFISPPARAMETERS, *PGFISPPARAMETERS;
```

See “GFISPAPI Parameter Data Structures” on page 549 for descriptions of each of these data structures.

GFISP Parameter Definitions

```
#define GFISP_FIXEDPOINTPARAMETER_FBITS             8
#define GFISP_FIXEDPOINTPARAMETER_BITS              16
#define GFISP_CONVERT_FLOAT_TO_FIXED_SIGNED(x)
          ((NvS16)(x * (2 << GFISP_FIXEDPOINTPARAMETER_FBITS)))
#define GFISP_CONVERT_FLOAT_TO_FIXED_UNSIGNED(x)
          ((NvU16)(x * (2 << GFISP_FIXEDPOINTPARAMETER_FBITS)))
#define GFISP_CONVERT_FIXED_TO_FLOAT(x)
          (((float)x) / (2 << GFISP_FIXEDPOINTPARAMETER_FBITS))
```

To simplify the interface, all fixed-point parameters are in `u8.8` format or 2's complement `s7.8` format regardless of internal precision. Generic macros have been provided to convert to and from floating point format on systems that support it. For actual range and precision of individual parameters, see the description of that parameter.

GFISPReadMWindowValues()

This function reads back the results of the four window measurements—M1, M2, M3, and M4. (Refer to "GFISP_PARA_M1WINDOW", "GFISP_PARA_M2WINDOW", "GFISP_PARA_M3WINDOW", and "GFISP_PARA_M4WINDOW" on pages 553 though 558.)

Because reading the value back from the hardware can only be done at frame intervals, it takes at least one frame to get the correct value.

The user may choose to call this function synchronously or asynchronously. When **`pEvent`** is NULL, this function returns synchronously until the hardware reading is finished. Otherwise, this function returns immediately and, when the values are correctly filled by the hardware, the GFISPAPI triggers **`pEvent`** to notify the caller that the data is ready.

Function Prototype

```
GF_RETTYPE GFISPReadMWindowValues(
    GF_HANDLE  IspHandle,
    GF_EVENT   pEvent,
    NvU32      szM1Buffer,
    NvU32      *pM1Buffer,
    NvU32      szM2Buffer,
    NvU32      *pM2Buffer,
    NvU32      szM3Buffer,
    NvU32      *pM3Buffer,
    NvU32      szM4Buffer,
    NvU32      *pM4Buffer );
```

Parameters

`IspHandle`	Handle specific to the GFISPAPI.
`szM1Buffer`	Maximum size of `M1Buffer` provided by caller (in bytes). The GFISPAPI fills the buffer as much as it can. If zero, the GFISPAPI doesn't fill the `M1Buffer`.
`*pM1Buffer`	Array of 8 data. First four 32-bit values stores average RGGB pixel value. Next four values for peak pixel value. Can be NULL, then the GFISPAPI doesn't fill M1Buffer.
`szM2Buffer`	Maximum size of `M2Buffer` provided by the caller (in bytes). The GFISPAPI fills the buffer as much as it can. If zero, the GFISPAPI doesn't fill `M2Buffer`.

Parameters (continued)

*pM2Buffer	Array of 4x4=16 data to store pixel count for each color component. The data is arranged as follows: {RCount1, GrCount1, GbCount1, BCount1, RCount2, GrCount2, GbCount2, BCount2, ...} Can be NULL and the GFISPAPI won't fill M2Buffer.
szM3Buffer	Maximum size of M3Buffer provided by the caller (in bytes). The GFISPAPI fills the buffer as much as it can. If zero, the GFISPAPI doesn't fill M3Buffer.
*pM3Buffer	Array to hold pixel samples. The size matches the M3 Window specification. The data is arranged as follows: {R1, R2, R3, R4}, {Gr1, Gr2, Gr3, Gr4}, {Gb1, Gb2, Gb3, Gb4}, {B1, B2, B3, B4}, {R5, R6, R7, R8}, ... Can be NULL and the GFISPAPI won't fill M3Buffer.
szM4Buffer	Maximum size of M4Buffer provided by the caller (in bytes). The GFISPAPI fills the buffer as much as it can. If zero, the GFISPAPI doesn't fill M4Buffer.
*pM4Buffer	Array of 2*M4WindowNumber value. Data is arranged as follows: {HF1, Luma1, HF2, Luma2, …} Can be NULL and the GFISPAPI won't fill M4Buffer.

Returns

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFISPAPI Parameter Data Structures

This section describes the following GFISPAPI data structures related to "GFISPGetParameter()" and "GFISPSetParameter()".

GFISP_PARA_TIMING

This data structure sets/retrieves image format and timing information.

An imager that supplies the pixel stream to the ISP controls the timing, and supplies the pixel clock, and horizontal (H) and vertical (V) sync timing pulses. The ISP is a slave to the imager, and locally regenerates line and frame timing based on the H and V input sync pulses.

Figure 12. illustrates the image format and timing.

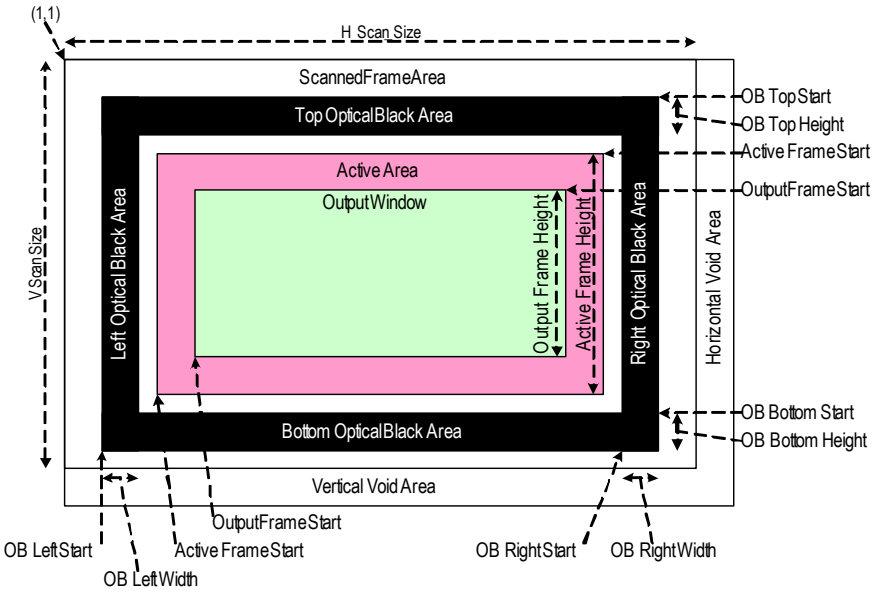


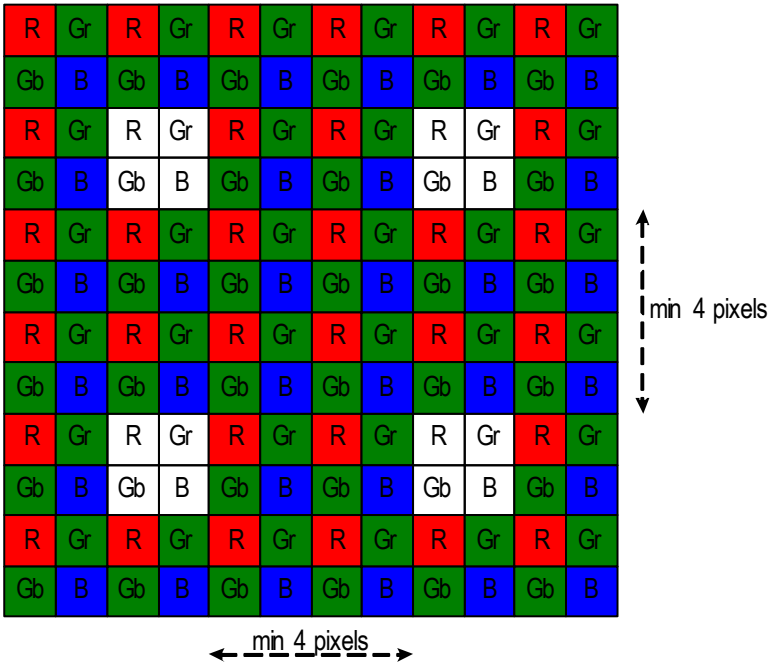
Figure 12. Timing Frame

The entire timing frame consists of scanned parts and idle parts (void periods between lines and frames). The overall scanned image frame is specified by **`hScanSize`**/**`vScanSize`** which includes H and V blank periods. Line and pixel counters are set to 1 at transitions of active edge of V and H sync input pulses. The coordinate for the entire frame starts with (1,1) instead of (0,0).

Active Area (**`ActiveRect`** in **`struct _GFISPIMGTIMING`**) represents the part which ISP processes.

Output Windows (**`OutputRect`** in **`struct _GFISPIMGTIMING`**) represents the output area from ISP. It is a subset of **`ActiveRect`**.

The pixel format is shown in.Figure 13.



Vertically, the minimum distance between any 2x2 bad pixel clusters is theoretically 2 pixels, because "corrected" bad pixels are stored into line buffers before being used in the demosaic process.

However, it is not ideal to use these "corrected" bad pixels when correcting other bad pixel clusters below it, therefore the minimum vertical distance is set to 4 pixels.

Figure 13. Pixel Format

Currently the SC15 supports only one signal format—the Bayer format from imagers. The image consists of alternating series of image lines—one with interleaved R and G, and another one with interleaved G and B. .

GFISP BAYER PATTERN Formats

```
typedef enum {
    GFISP_RGGB = 0,
    GFISP_GRBG = 1,
    GFISP_GBRG = 2,
    GFISP_BGGR = 3
} GFISP_BAYERPATTERN_SEL;
```

GFISPIMGTIMING Structure

```
typedef struct _GFISPIMGTIMING {
    NvU16 hScanSize,
          vScanSize;
    NvU8  hSyncEdge,
          vSyncEdge;
    GFISP_BAYERPATTERN_SEL bayerSel;

    GFRECT  ActiveRect;
    GFRECT  OutputRect;
} GFISPIMGTIMING, *PGFISPIMGTIMING;
```

GFISPIMGTIMING Fields

`hScanSize`	Number of pixel clock periods per scan line (including H blanking).
`vScanSize`	Number of line periods per scan frame (including V blanking).
`hSyncEdge`	Active edge selection.
`vSyncEdge`	0 = positive transition, 1 = negative transition.
`bayerSel`	Bayer format variation selection.
`ActiveRect`	Active image area.
`OutputRect`	Output image position and size.

GFISP_PARA_M1WINDOW

The GFISPAPI provides four methods that programmers can use to gather statistics on pixel values. Based on the values, the programmer can take appropriate compensating action as needed. The four methods are M1 Window, M2 Window, M3 Window, and M4 Window.

This structure describes the M1 Window method, which measures average/peak pixel values inside the specified area. It returns eight values—four values (RGGB) represent average pixels, and four values represent peak pixels. The measurement result can be used for white balance, either automatically by AWB or manually by the programmer.

Refer to the function "GFISPReadMWindowValues()" for reading the measured values.

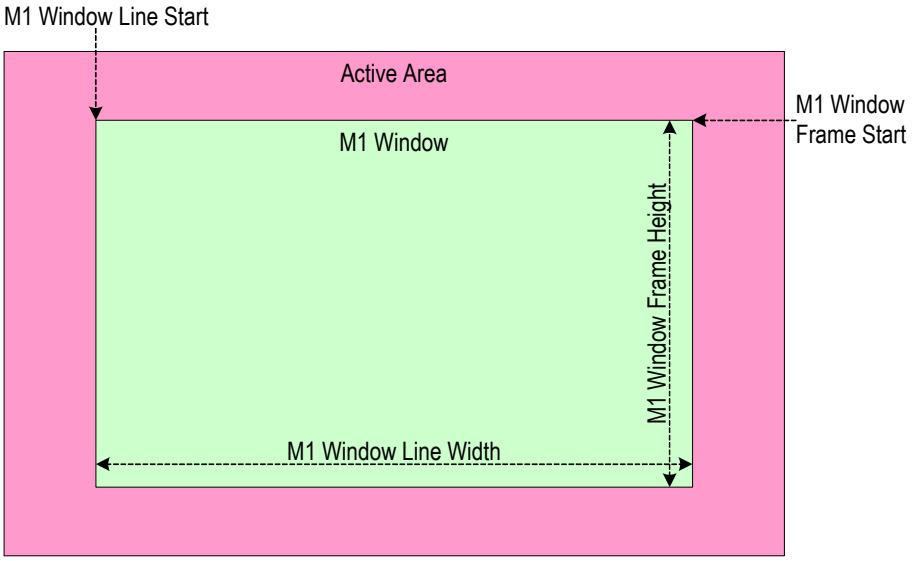


Figure 14. M1 Window Measurement Method

GFISP M1 Sample Densities

```
typedef enum {
    M1SAMPLE32 = 0,
    M1SAMPLE64 = 1
} M1_SAMPLE_SEL;
```

GFISPM1WINDOW Structure

```
typedef struct _GFISPM1WINDOW {
    GFRECT          M1WindowRect;
    M1_SAMPLE_SEL   horzSampSel,
                    vertSampSel;
    NvU16           PeakMax;
    NvU16           TemporalFilterStrength
} GFISPM1WINDOW, *PGFISPM1WINDOW;
```

GFISPM1WINDOW Fields

`M1WindowRect`	M1 Window size/position.
`horzSampSel`	Horizontal sample density.
`vertSampSel`	Vertical sample density.
`PeakMax`	Maximum limit of peak detection.
`TemporalFilter Strength`	Selects temporal filter coefficients. Valid range is [0, 7]. Value 0 effectively turns off temporal filter.

GFISP_PARA_M2WINDOW

This structure describes the M2 Window method for gathering statistics on pixel values. The M2 Window method measures the number of pixels at specified ranges inside a specified area. A histogram with four bins is generated.

- Refer to the function "GFISPReadMWindowValues()" for reading the measured values.

- This parameter is associated with the attribute **GFISP_ATTR_ENABLE_M2**. (See "GFISPATTRIBUTES Enumeration Type" on page 542.)

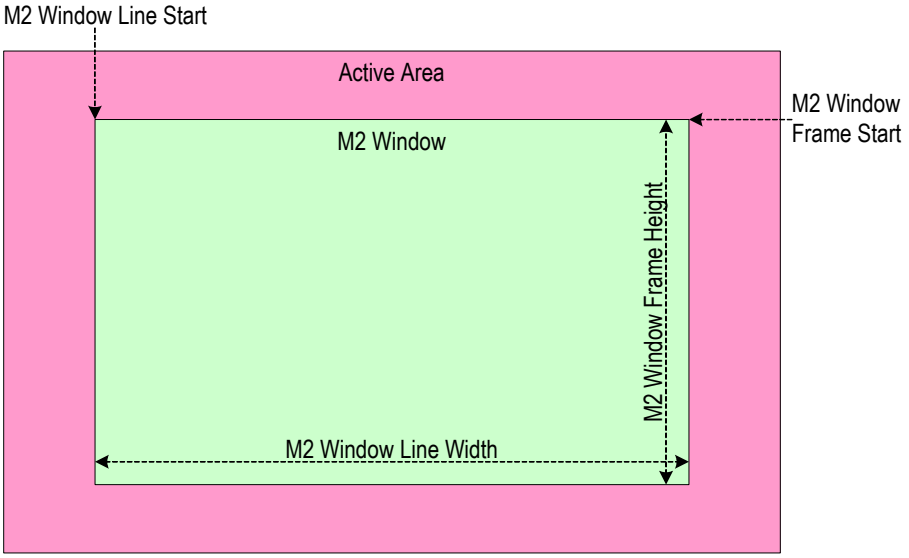


Figure 15. M2 Window Measurement Method

GFISPM2WINDOW Structure

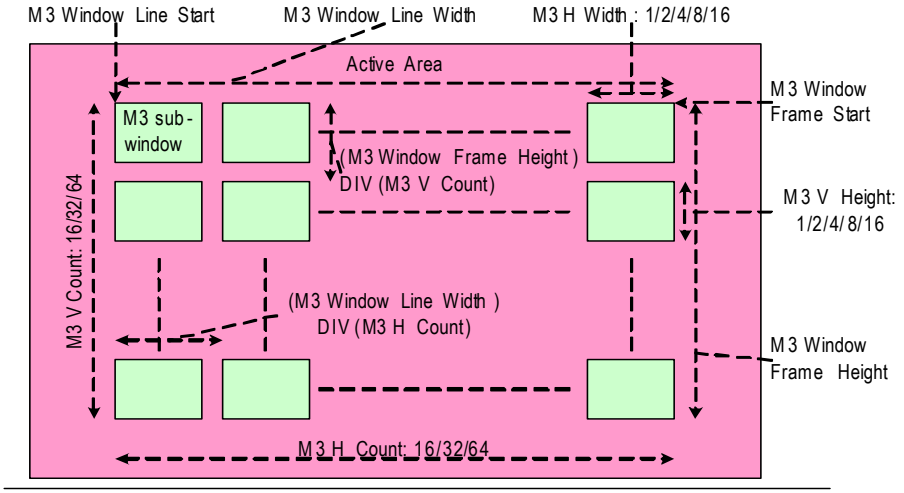
```
typedef struct _GFISPM2WINDOW {
    GFRECT M2WindowRect;
    NvU16  HistBinL1,
           HistBinL2,
           HistBinL3;
    NvU16  reserved;
} GFISPM2WINDOW, *PGFISPM2WINDOW;
```

GFISPM2WINDOW Fields

M2WindowRect	M2 Window size/position.
HistBinL1	Cut point of 1st bin.
HistBinL2	Cut point of 2nd bin.
HistBinL3	Cut point of 3rd bin.

GFISP_PARA_M3WINDOW

This structure describes the M3 Window method for gathering statistics on pixel values, The M3 Window method generates a group of pixel samples from the specified area. Each sub-window generates one 32-bit value.



Notes:

a. The horizontal interval of M3 sub-windows are derived from M3 Window Line Width divided by NHWINNUM, and rounded down to the nearest integer. This is essentially removing the 4/5/6 least significant bits from M3 Window Line Width, depending on 16/32/64 NHWINNUM,respectively.

b. The vertical interval of M3 sub-windows are derived from M3 Window Frame Height divided by NHWINNUM, and rounded down to the nearest integer. This is essentially removing the 4/5/6 least significant bits from M3 Window Frame Height, depending on 16/32/64 NHWINNUM,respectively.

Figure 16. M3 Window Measure Method

- Refer to the function "GFISPReadMWindowValues()" for reading the measured values.
- This parameter is associated with the attribute **GFISP_ATTR_ENABLE_M3**. (See "GFISPATTRIBUTES Enumeration Type" on page 542.)

GFISPM3WINDOW Structure

```
typedef struct _GFISPM3WINDOW {
    GFRECT M3WindowRect;
    NvU8   SubWindowCountH,
```

GFISPM3WINDOW Structure

```
            SubWindowCountV;
    NvU16   SubWindowWidth,
            SubWindowHeight;
    NvU16   reserved;
} GFISPM3WINDOW, *PGFISPM3WINDOW;
```

GFISPM3WINDOW Fields

`M3WindowRect`	M3 Window size/position.
`SubWindowCountH`	Sub-window number in horizontal direction. Can be {16,32,64}.
`SubWindowCountH`	Sub-window number in vertical direction. Can be {16,32,64}.
`SubWindowWidth`	Width of sub-window. Can be {2,4,8,16,32}.
`SubWindowHeight`	Height of sub-window. Can be {2,4,8,16,32}.

GFISP_PARA_M4WINDOW

This structure describes the M4 Window method for gathering statistics on pixel values. The M4 Window method takes statistical measurements for auto-focus control. It generates two values for each sub-window. One value represents the high frequency factor inside the sub-window. The other value represents sub-window luminance.

- Refer to the function "GFISPReadMWindowValues()"for reading the measured values.
- This parameter is associated with the attribute **`GFISP_ATTR_ENABLE_M4`**. (See "GFISPATTRIBUTES Enumeration Type" on page 542.)

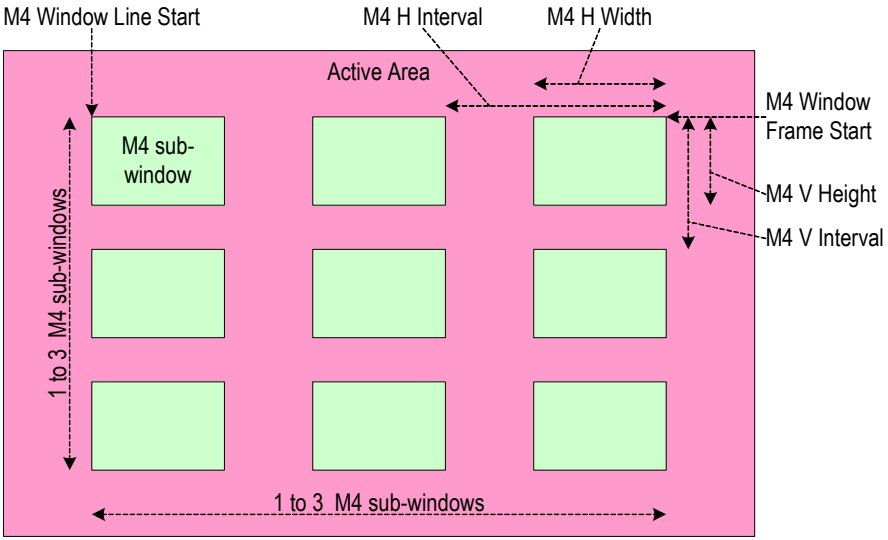


Figure 17. M4 Window Measure Method

There are {1, 2, 3, 4, 6, 9} sub-windows in {1,2,3}x{1,2,3} configurations.

The top-left corner of the first M4 sub-window is specified by *Top* and *Left*.

- **`SubWindowCountH`**/**`HSubWindowCountV`** independently specifies the number of sub-windows in horizontal and vertical directions.
- **`SubWindowIntervalH`**/**`HSubWindowIntervalV`** specifies vertical and horizontal intervals of sub-windows.

- **SubWindowWidth**/**SubWindowHeight** specifies the size of a sub-window.

SC15 implementation has the following constraints

- **SubWindowintervalH** >= **SubWindowWidth**
- **SubWindowintervalV** >= **SubWindowHeight**

GFISPM4WINDOW Structure

```
typedef struct _GFISPM4WINDOW {
    NvU16  Left, Top;
    NvU8   SubWindowCountH, HSubWindowCountV;
    NvU16  SubWindowWidth, SubWindowHeight;
    NvU16  SubWindowIntervalH, SubWindowIntervalV;
    NvU8   CoreLimit;
    NvU8   reserved;
} GFISPM4WINDOW, *PGFISPM4WINDOW;
```

GFISPM4WINDOW Fields

`Top`	Top of the first M4 sub-window.
`Left`	Left of the first M4 sub-window.
`SubWindowCountH`	Sub-window number in horizontal direction. Can be {1,2,3}.
`SubWindowCountV`	Sub-window number in vertical direction. Can be {1,2,3}.
`SubWindowWidth`	Width of the sub-window.
`SubWindowHeight`	Height of the sub-window.
`SubWindowIntervalH`	Interval of sub-windows in horizontal direction.
`SubWindowIntervalV`	Interval of sub-windows in vertical direction.
`CoreLimit`	Top limit of noise coring for M4 statistics gathering.

GFISP_PARA_OB

The very first ISP operation to be applied to the input pixel signal is to establish “black” level. This anchors signal activities to a black reference level, or “zero” level.

There are two methods of accomplishing optical blacking. One method is performed on each frame and is called Frame-based Optical Blacking (FOB). The other method is performed on each scan line and is called Line-based Optical Blacking (LOB).

Under FOB, optical black level can be established in one of two ways—by setting it manually, or more commonly by letting the ISP measure it automatically. In the automatic method the user can specify black regions which surround the active imaging area. The pixel levels from these regions are used to determine the input signal level of reference black, which is mapped to “zero” level in the subsequent pixel processing.

LOB is applied to each scan line and is used to detect and compensate for fluctuations in the imager input signal level of reference black.

The user can choose to use either FOB, or LOB, or both.

This parameter is associated with the following attributes (see “GFISPATTRIBUTES Enumeration Type” on page 542):

- **GFISP_ATTR_ENABLE_FOB**
- **GFISP_ATTR_ENABLE_FOB_SELFRESET**
- **GFISP_ATTR_ENABLE_FOB_TEMPORAL**
- **GFISP_ATTR_ENABLE_MANUALOB**

- **GFISP_ATTR_ENABLE_LOB**

GFISPOBCONTROL Structure

```
typedef struct _GFISPOBCONTROL {
    /* Four OB Regions
       If ISP's OB capability is GFISP_CAP_THREE_OB_REGIONS,
       then six of the 16 OB coordinates are used:
       OBTopRegionStartV,  OBTopRegionHeight,
       OBLeftRegionStartH, OBLeftRegionWidth
       OBBotomRegionStarttV, OBBottomRegionHeight

       Right OB region is not supported.
       Top region width is assumed equal to input width.
       Left region height is assumed equal to input height.
       Bottom region width is assumed equal to input width.
       Bottom region is only used if
       GFISP_ATTR_ENABLE_FOB_SELFRESET is disabled.

       If ISP's OB capability is GFISP_CAP_FOUR_OB_REGIONS,
       then all sixteen OB coordinates are used.
    */
    NvU32 OBTopRegionStartV,    OBTopRegionHeight,
          OBTopRegionStartH,    OBTopRegionWidth;
    NvU32 OBBottomRegionStartV, OBBottomRegionHeight,
          OBBottomRegionStartH, OBBottomRegionWidth;
    NvU32 OBLeftRegionStartV,   OBLeftRegionHeight,
          OBLeftRegionStartH,   OBLeftRegionWidth;
    NvU32 OBRightRegionStartV,  OBRightRegionHeight,
          OBRightRegionStartH,  OBRightRegionWidth;
    /* Frame OB */
    NvU8 FOBCoef;
    // Filter coefficient of F-OB acquisition.
    NvU8 TOBCoef;
    // Filter coefficient of F-OB temporal filter.
    // Valid range is [0-7].
    /* Line OB */
    NvU8 LOBWidth;
    // Width used for L-OB. Valid values are {2, 4, 8}.
    /* Manual OB */
    NvU8 FOBAdjust;
    // Manual adjustment applied to the black level.
} GFISPOBCONTROL, *PGFISPOBCONTROL;
```

GFISPOBCONTROL Fields

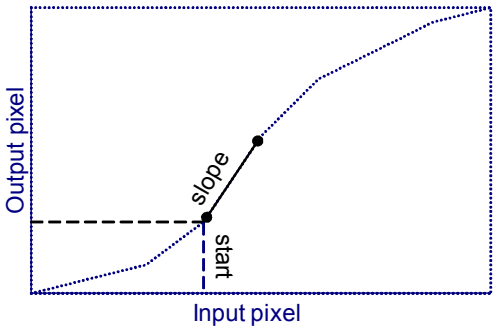
`OBTopRegionStart` `OBTopRegionHeight`	Top FOB region.
`OBBottomRegionStart` `OBBottomRegionHeight`	Bottom FOB region.
`OBLeftRegionStart` `OBLeftRegionWidth`	Left FOB region.
`OBRightRegionStart` `OBRightRegionWidth`	Right FOB region.
`FOBCoef`	Filter coefficient of FOB acquisition inside one frame. It represents how strong the last OB pixel factors against previous pixels. Valid range is [0,15].
`TOBCoef`	Filter coefficient of FOB temporal filter. It represents how strong current frame OB factors against previous frames. Valid range is [0,15].
`FOBAdjust`	Manual adjustment applied to the black level. Only effective when GFISP_PARA_MANUALOB is set.
`LOBWidth`	This is the only variable related to LOB. Number of columns sampled for smoothing. Valid values are {2,4,8}.

GFISP_PARA_DEKNEE

Imagers tend to show some degree of non-linearity in their light-to-electric transfer characteristics. DeKnee unit attempts to improve the linearity.

NVIDIA ISP implements DeKnee as a piece-wise linear approximation of up to 16 segments.

This structure describes the start and slope values for the DeKnee look-up table. The caller may choose to divide the table into 4, 8, or 16 linear segments (4, 8, or 16 entries in the table).



GFISPDEKNEEENTRY Structure

```
typedef struct _GFISPDEKNEEENTRY {
    NvU16  start;
    NvU16  slope;
} GFISPDEKNEEENTRY, *PGFISPDEKNEEENTRY;
```

This structure describes one approximated linear segment

GFISPDEKNEEENTRY Fields

`start`	Start of a range.
`slope`	Slope of the linear. Unsigned fixed-point format. Valid range is [`0, 3 63/64`] with precision of `1/64`.

GFISPDEKNEECONTROL Structure

```
typedef struct _GFISPDEKNEECONTROL {
    NvU32              nEntries;
    GFISPDEKNEEENTRY   pEntries[];
} GFISPDEKNEECONTROL, *PGFISPDEKNEECONTROL;
```

GFISPDEKNEECONTROL Fields

`nEntries`	Number of entries in the `pEntry` table. Valid values are {4,8,16}.
`pEntries`	DeKnee entry table.

GFISP_PARA_LENSSHADING

An optical lens in front of the imager tends to introduce shading and unevenness in the strength of incident light. It is strongest at the center and diminishes as the distance from the center increases. Actual shading characteristics vary depending on the choice of optics.

This structure describes a method for compensating for this shading by applying varying levels of gain to the pixel signal in the reverse function of the shading.

The GFISPAPI implements this as a second degree quadratic equation for each color component at H and V:

```
F = 1 + X * (Coef1 + Coef2 *X)
```

GFISPLENSSHADING Structure

```
typedef struct _GFISPLENSSHADING {
    NvU16  hCenter, vCenter;
    NvU16  rHorzCoef1, rHorzCoef2;
    NvU16  rVertCoef1, rVertCoef2;
    NvU16  gHorzCoef1, gHorzCoef2;
    NvU16  gVertCoef1, gVertCoef2;
    NvU16  bHorzCoef1, bHorzCoef2;
    NvU16  bVertCoef1, bVertCoef2;
} GFISPLENSSHADING, *PGFISPLENSSHADING;
```

GFISPLENSSHADING Fields

`hCenter` `vCenter`	Center point of camera lens.

The coefficients below are unsigned fixed-point format.
Valid range is [0, 7 63/64] with precision of 1/64.

`rHorzCoef1` `rHorzCoef2`	First/second order coefficient for Horizontal Red.
`rVertCoef1` `rVertCoef2`	First/second order coefficient for Vertical Red.

GFISPLENSSHADING Fields (continued)

gHorzCoef1 gHorzCoef2	First/second order coefficient for Horizontal Green.
gVertCoef1 gVertCoef2	First/second order coefficient for Vertical Green.
bHorzCoef1 bHorzCoef2	First/second order coefficient for Horizontal Blue.
bVertCoef1 bVertCoef2	First/second order coefficient for Vertical Blue.

GFISP_PARA_WB

This structure controls the white balance method.

White Values

RGB component signal levels are balanced to render white objects as white. This is a normalization step of electronic signal of RGB color primaries.

Depending on the “color temperature” of the white reference, the relative strength of RGB tri-stimulus values vary. In the electric signal, ER/EG/EB corresponding to the RGB tri-stimulus values are normalized so that white (of 100% strength) is represented by ER = EG = EB = 1. The original image in front of the imager is reproduced only when a display device does precise reverse conversion, which is a purely ideal case. Otherwise, the original image is rendered in different shades of colors depending on the reference white for which the display device is tuned.

Gray Values

RGB pixel values that represent “gray” objects may be obtained in various ways. For example, a picture frame can be filled with a “white” object, or average values for a large number of pixels can be calculated, or a set of average values collected in many sampling spotscan be used to represent the approximated “gray” object.

Gain Factors and White Balance

Digital gain values are dedicated to each of the four color channels (R, Gr, Gb, B).

Depending on how to set gain factors, there are two ways to use White Balance. A commonly used method is called Dynamic WB, or AWB. Under AWB, gain values are updated automatically by ISP internal AWB control based on the average values and peak values in {R, G1, G2, B} color channels (Refer to "GFISPGetParameter()" with the parameter "GFISP_PARA_M1WINDOW".)

- If the frame exposure is high ("Top" level, controlled by **`ThresholdT2M`**/**`ThresholdM2T`**), gains are calculated from the *average* RGrGbB values.
- If the frame exposure is low ("Bottom" level, controlled by **`ThresholdM2B`**/**`ThresholdB2M`**), gains are calculated from *peak* RGrGbB values.
- If the frame exposure is in the middle range, gains are calculated from *both* average and peak values.

Under AWB, the user can send a command to freeze the current gain value by setting **`GFISP_ATTR_AWB_HOLD`** to 1. Then the current AWB gain value is applied to all future frames.

The other method is called pre-programmed WB or Static WB. It sets pre-programmed gain values and applies them all time. Under this mode, the programmer determines the proper gain values based on the M1/M3 measurement window output.

This parameter is associated with the attributes **`GFISP_ATTR_ENABLE_WB`**, **`GFISP_ATTR_ENABLE_AWB`**, and **`GFISP_ATTR_AWB_HOLD`**. (See "GFISPATTRIBUTES Enumeration Type" on page 542.)

GFISPWBCONTROL Structure

```
typedef struct _GFISPWBCONTROL {
    NvU16  RGain,
           GrGain,
           BGain,
           GbGain;

    NvU16   TopClipLimit;
    NvU16   PeakRef;
    NvU16   RGainMax, RGainMin;
    NvU16   GGainMax, GGainMin;
    NvU16   BGainMax, BGainMin;

    NvU16   ThresholdT2M,
```

GFISPWBCONTROL Structure (continued)

```
            ThresholdM2T,
            ThresholdM2B,
            ThresholdB2M;

} GFISPWBCONTROL, *PGFISPWBCONTROL;
```

GFISPWBCONTROL Fields

The following data member affects both WB control and AWB control.

`TopClipLimit`	Lower limit of gain-based clipping. Value 0 effectively turns off Top-clipping.

These four data members belong to Static WB control, and are effective only when `GFISP_ATTR_ENABLE_AWB==0`.

`RGain`	Gain factor for Red.
`GrGain`	Gain factor for Green on Red.
`BGain`	Gain factor for Blue.
`GbGain`	Gain factor for Green on Blue.

The remaining data members belong to AWB control, and are effective only when `GFISP_ATTR_ENABLE_AWB==1`.

`PeakRef`	White reference level for gain adjustment.
`RGainMax`	Max Gain factor for Red.
`RGainMin`	Min Gain factor for Red.
`GGainMax`	Max Gain factor for Green.
`GGainMin`	Min Gain factor for Green.
`BGainMax`	Max Gain factor for Blue.
`BGainMin`	Min Gain factor for Blue.
`ThresholdT2M`	Top-to-Middle transition level.
`ThresholdM2T`	Middle-to-Top transition level.
`ThresholdM2B`	Middle-to-Bottom transition level.
`ThresholdB2M`	Bottom-to-Middle transition level.

GFISP_PARA_DEMOSAIC

This structure controls the demosaic process, which uses 2D interpolation to convert Bayer format R/G/B pixel signals obtained from the imager to generate R/G/B component values.

Under the SC15 hardware implementation, the Demosaic unit is built with bad pixel (BP) and noise reduction (NR) control inside. So picking demosaic mode affects bad pixel and noise reduction control. Unless limited by memory bandwidth, programmers should always pick BPNR modes. Otherwise, BP and NR features are not available, even if you try to turn on BP or NR controls with **`GFISPSetAttribute()`**.

GFISP Demosaic Modes

```
typedef enum {
    GFISP_DM3X3 = 0,          // 3x3 one-shot demosaicing.
    GFISP_DM3X3BPNR = 1,
        // 3x3 demosaicing/bad-pixel/noise-reduction.
    GFISP_DM5X5 = 2,          // 5x5 one-shot demosaicing.
    GFISP_DM5X5BP = 3,        // 5x5 demosaicing/bad-pixel.
    GFISP_DM5X5BPNR = 4
        // 5x5 demosaicing/bad-pixel/noise-reduction.
} GFISP_DEMOSAIC_MODE_SEL;
```

GFISPDEMOSAICCONTROL Structure

```
typedef struct _GFISPDEMOSAICCONTROL {
    GFISP_DEMOSAIC_MODE_SEL  DemosaicMode;
} GFISPDEMOSAICCONTROL, *PGFISPDEMOSAICCONTROL;
```

GFISPDEMOSAICCONTROL Fields

`DemosaicMode`	3x3 or 5x5 demosaic mode.

GFISP_PARA_EDGEENHANCE

This structure controls the edge enhancement process, accomplished by the following:

13. Extracting the high frequency component (HF) of the pixel
14. Amplifying HF if the extracted value is “large” (that is, coring)
15. Adding the amplified HF back to the Y channel of the pixel.

In the SC15 implementation, edge enhancement control resides in the Demosaic unit. It is effective only if **DM3X3BPNR**, **DM5X5NR**, or **DM5X5BPNR** demosaic mode is selected. The threshold that's used for coring is not a pre-determined fixed value, but instead is adaptive depending on the brightness level. If the pixel surround is bright (more visible), the coring threshold will be larger, which means only stronger edges are enhanced. Weaker edges are considered more noise-prone and therefore are not enhanced.

Two control parameters are used to determine the adaptive coring threshold:

- **GCoreLimit**. Minimum coring level. The coring level increases when the pixel’s surrounding brightness level goes up.
- **GCoreScaleSel**. Influence of the brightness level to the coring level. The higher the **GCoreScaleSel** is, the larger the coring level grows under the same brightness.

GFISP GCoring Limit Scaling Factors

```
typedef enum {
    GFISP_GCLIMIT_SCALE_4 = 0,          // Range [1,4.5]
    GFISP_GCLIMIT_SCALE_8 = 1,          // Range [1,8.5]
    GFISP_GCLIMIT_SCALE_16= 2,          // Range [1,16.5]
    GFISP_GCLIMIT_SCALE_32= 3           // Range [1,32.5]
} GFISP_GCORINGLIMIT_SCALE_SEL;
```

GFISPEDGEENHANCECONTROL Structure

```
typedef struct _GFISPEDGEENHANCECONTROL {
    NvU16  StrengthLevel;
    NvU16  GCoringLimit;
    GFISP_GCORINGLIMIT_SCALE_SEL  GCoringScaleSel;
} GFISPEDGEENHANCECONTROL, *PGFISPEDGEENHANCECONTROL;
```

GFISPEDGEENHANCECONTROL Fields

`StrengthLevel`	How heavily the edge enhancement is applied. Unsigned fixed-point format. Valid range is [0, 3 7/8] with a precision of 1/8, plus [4, 15 1/2] with a precision of 1/2.
`GCoringLimit`	Minimum coring level.
`GCoringScaleSel`	Scaling factor of `GCoringLimit`.

GFISP_PARA_NOISECONTROL1

This structure controls the noise control method 1 process.

There are two noise reduction functions in the ISP data path:

- 3D noise reduction
- False color reduction

`GFISP_PARA_NOISECONTROL1` applies to the first method. It also resides in the Demosaic unit along with edge enhancement. It is effective only if **`DM3X3BPNR`**, **`DM5X5NR`**, or **`DM5X5BPNR`** demosaic mode is selected.

This method reduces noise not only in the image plane but also in the intensity domain. In other words, low pass filtering is applied to pixels that have a similar pixel value and are also close to the same location. **`FilterStrengthLevel`** defines the filter kernel size on the intensity domain. Larger **`FilterStrengthLevel`** generates a smoother but blurry image.

GFISPNOISECONTROL1 Structure

```
typedef struct _GFISPNOISECONTROL1 {
    NvU16  FilterStrengthLevel;
} GFISPNOISECONTROL1, *PGFISPNOISECONTROL1;
```

GFISPNOISECONTROL1 Fields

`FilterStrengthLevel`	Weighted average scaling factor. Valid range is [0,7]

GFISP_PARA_NOISECONTROL2

This structure controls the noise control method 2 process.

There are two noise reduction functions in the ISP data path:

- 3D noise reduction
- False color reduction

GFISP_PARA_NOISECONTROL2 applies to the second method. It resides in the color correction unit.

This method reduces the colorfulness of pixels at shadow region or at edges, to de-emphasize color noise and color aliasing effects. **HighLimit** and **TransitionWidth** specify the pixels that will undergo the color reduction. If the luminance level of the pixel is larger than (**HighLimit** + **TransitionWidth**), the pixel's colorfulness is intact. If the luminance level is smaller than **HighLimit**, the colorfulness is reduced to a minimum (i.e., no color correction is applied). Pixels with luminance levels between these limits pass through an intermediate colorfulness reduction factor. **CCScalFactor** specifies how much color reduction should be applied at edge pixels. Larger **CCScalFactor** creates a more noticeable color reduction at the edges.

GFISPNOISECONTROL2 Structure

```
typedef struct _GFISPNOISECONTROL2 {
    NvU16  HighLimit;
    NvU16  TransitionWidth;
    NvU16  CCScaleFactor;
} GFISPNOISECONTROL2, *PGFISPNOISECONTROL2;
```

GFISPNOISECONTROL2 Fields

HighLimit	Upper-limit level of noise reduction curve
TransitionWidth	Width of noise reduction curve. Valid values are {16, 32, 64, 128}
CCScaleFactor	Slope of noise reduction curve. Unsigned fixed-point format. Valid range is [0, 31].

GFISP_PARA_BADPIXEL

This structure controls a method to conceal bad pixels.

The process also resides in the Demosaic unit along with edge enhancement and noise control 1. It is effective only if **DM3X3BPNR** or **DM5X5BPNR** demosaic mode is selected.

If a pixel value deviates greatly from its surrounding pixels, the pixel is considered to be a bad pixel. The deviation is defined in terms of the percentage of the average value of the surrounding pixels.

Two ratio coefficients are used:

- If the pixel is within a flat area, **LowCoef** is used,
- If the pixel is at an edge region, **UpCoef** is used.

If the average of the surrounding pixels is lower than **DarkLevel**, it is considered to be at a shadow region, and **DarkLevel** instead of the average value will be used to calculate the deviation threshold.

GFISPBADPIXELCONTROL Structure

```
typedef struct _GFISPBADPIXELCONTROL {
    NvU16  DarkLevel;
    NvU16  UpCoef, LowCoef;
} GFISPBADPIXELCONTROL, *PGFISPBADPIXELCONTROL;
```

GFISPBADPIXELCONTROL Fields

DarkLevel	Bad-pixel dark level.
UpCoef	Coefficient for Upper threshold. Unsigned fixed-point format. Valid range is [0, 1 7/8] with precision of 1/8.
LowCoef	Coefficient for Lower threshold. Unsigned fixed-point format. Valid range is [0, 1 7/8] with precision of 1/8.

GFISP_PARA_COLORCORRECT

This structure controls the color correction process.

In a strict sense, the color correction module transforms a given color space (based on the optical characteristics of the imager) to a specific color space such as described by the CCIR 601 standard. The color component signals are then handled according to that standard color space from that point on.

For high-end cameras, color correction is used to match some sensitive colors such as flesh tone to a desired shade of colors, and is handled primarily by experts who know what they are doing.

In practice, there may be a simpler means with fewer control parameters, where the color shade of images is adjusted to suit the viewer's taste. So, gain and offset of R, G, and B pixels are individually adjusted.

GFISP color correction consists of 3x3 matrix factors for the camera RGB (cRGB) to standard RGB (sRGB) conversion. Subjective adjustment of color can also be included.

GFISPCOLORCORRECTION Structure

```
typedef struct _GFISPCOLORCORRECTION {
    NvS32  R2R, R2G, R2B;
    NvS32  G2R, G2G, G2B;
    NvS32  B2R, B2G, B2B;
} GFISPCOLORCORRECTION, *PGFISPCOLORCORRECTION;
```

GFISPCOLORCORRECTION Fields

`R2R R2G R2B`
`G2R G2G G2B`
`B2R B2G B2B`

3x3 Color conversion matrix.

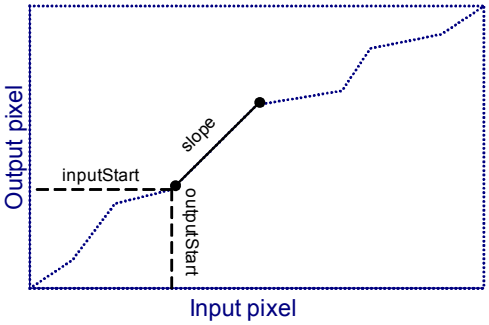
- Coefficients in the main diagonal (`R2R`, `G2G`, and `B2B`): Unsigned. Valid range is [0, 7 255/256].
- Off-diagonal coefficients: Signed. Valid range is [-7 255/256, 7 255/256].

GFISP_PARA_GAMMACORRECT

This structure controls the gamma correction process.

Gamma correction applies non-linear characteristics to compensate for the non-linear characteristics of display devices such as cathode ray tubes driven by R, G, B primary color signals. So, in a strict sense, this must be applied to R, G, B signals.

In SC15 implementation, gamma correction is approximated using up to 16 line segments. Each line entry must be provided in ascending order (within the input range).



GFISPGAMMAENTRY Structure

```
typedef struct _GFISPGAMMAENTRY {
    NvU16  inputStart;
    NvU16  outputStart;
    NvS16  slope;
    NvU16  reserved;
} GFISPGAMMAENTRY, *PGFISPGAMMAENTRY;
```

GFISPGAMMAENTRY Fields

`inputStart`	Input start value of line segment.
`outputStart`	Output start value of line segment.
`slope`	Slope of the line segment. Signed fixed-point format. Valid range is [-7 63/64, 7 63/64].

GFISPGAMMACONTROL Structure

```
typedef struct _GFISPGAMMACONTROL {
    NvU32            nEntries;
```

GFISPGAMMACONTROL Structure (continued)

```
    GFISPGAMMAENTRY  pEntries[];
} GFISPGAMMACONTROL, *PGFISPGAMMACONTROL;
```

GFISPGAMMACONTROL Fields

nEntries	Number of entries in the pEntries table. Valid range is [1, 16].
pEntries	Gamma entry table.

GFISP_PARA_YUVCONVERT

This structure controls the YUV conversion process, which converts RGB color space to corresponding YUV color space, based on the following 3x3 matrix:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} := \begin{bmatrix} ary & agy & aby \\ aru & agu & abu \\ arv & agv & abv \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

{Y, U, V} shall have offset values that are not shown in the formula above.

GFISP YUV Output Signal Ranges

```
/*
    Signal Range of YUV Outputs
*/
typedef enum {
    YRANGE_FULL     = 0,       // [0,255]
    YRANGE_NEARFULL = 1,       // [1,254] exclude 00 and FF
    YRANGE_ITU601   = 2        // ITU601 standard
} YRANGE_SEL;
```

GFISPYUVCONVERSION Structure

```
typedef struct _GFISPYUVCONVERSION {
    YRANGE_SEL  YRangeSel;
    NvS16       YOff;
    NvS16       R2Y, G2Y, B2Y;
    NvS16       R2U, G2U, B2U;
    NvS16       R2V, G2V, B2V;
} GFISPYUVCONVERSION, *PGFISPYUVCONVERSION;
```

GFISPYUVCONVERSION Fields

YrangeSel	YUV output signal range.
YOff	Y value offset. Valid range is [-128, 127].
R2Y G2Y B2Y R2U G2U B2U R2V G2V B2V	3x3 YUV conversion matrix. • G2Y is unsigned fixed-point format. Valid range is [0, 1 255/256]. • R2Y and B2Y are unsigned fixed-point format. Valid range is [0, 255/256]. • R2U, G2U, B2U, R2V, G2V, and B2V are signed fixed-point format. Valid range is [-255/256, 255/256].

GFISP Attribute and Parameter Associations

Table 1 lists the parameters that are associated with the GFISP attributes.

Table 1 GFISP Attribute and Parameter Associations

Attribute	Associated Parameters
GFISP_ATTR_ENABLE_ISP	All
GFISP_ATTR_ENABLE_FOB	GFISP_PARA_OB
GFISP_ATTR_ENABLE_FOB_SELFRESET	GFISP_PARA_OB
GFISP_ATTR_ENABLE_FOB_TEMPORAL	GFISP_PARA_OB
GFISP_ATTR_ENABLE_LOB	GFISP_PARA_OB
GFISP_ATTR_ENABLE_DEKNEE	GFISP_PARA_DEKNEE
GFISP_ATTR_ENABLE_LENS_SHADING	GFISP_PARA_LENSSHADING
GFISP_ATTR_ENABLE_WB	GFISP_PARA_WB
GFISP_ATTR_ENABLE_AWB	GFISP_PARA_WB
GFISP_ATTR_AWB_HOLD	GFISP_PARA_WB
GFISP_ATTR_ENABLE_DEMOSAIC	GFISP_PARA_DEMOSAIC
GFISP_ATTR_ENABLE_EDGE_ENHANCE	GFISP_PARA_EDGEENHANCE
GFISP_ATTR_ENABLE_NOISE_CONTROL1	GFISP_PARA_NOISECONTROL1, GFISP_PARA_DEMOSAIC
GFISP_ATTR_ENABLE_BAD_PIXEL	GFISP_PARA_BADPIXEL, GFISP_PARA_DEMOSAIC
GFISP_ATTR_ENABLE_COLOR_CORRECTION	GFISP_PARA_COLORCORRECT
GFISP_ATTR_ENABLE_NOISE_CONTROL2	GFISP_PARA_NOISECONTROL2

Table 1 GFISP Attribute and Parameter Associations

Attribute	Associated Parameters
GFISP_ATTR_ENABLE_GAMMA_CORRECTION	GFISP_PARA_GAMMACORRECT
GFISP_ATTR_ENABLE_YUV_CONVERSION	GFISP_PARA_YUVCONVERT
GFISP_ATTR_ENABLE_NEGATIVE_EFFECT	
GFISP_ATTR_ENABLE_M2	GFISP_PARA_M2WINDOW
GFISP_ATTR_ENABLE_M3	GFISP_PARA_M3WINDOW
GFISP_ATTR_ENABLE_M4	GFISP_PARA_M4WINDOW
GFISP_ATTR_ENABLE_STATS_RAISE	
GFISP_ATTR_ENABLE_STATS_INT	
GFISP_ATTR_STATS_LINE_END	

CHAPTER

20

Resource Manager Read DMA (GFRmRDMA)

GFRmRDMA Overview

Read DMA (RDMA) supports rectangular and non-rectangular reads.

For rectangular reads, the software must set up the stride, the width, and the number of lines. This information is needed to describe a buffer. In the case of non-rectangular reads, a buffer header to describe the buffer is expected. In both cases you must specify the number of buffers.

Tip: If an engine has already generated the buffer, then use non-rectangular read. Otherwise, use rectangular read.

GFRmRDMA Reference

The GFRmRDMA API consists of

- ”GFRmRDMA Functions’
- ”GFRmRDMA Structures’
- ”GFRmRDMA Enumerations’
- ”GFRmRDMA Flags’

GFRmRDMA Functions

This section describes the following GFRmRDMA functions:

GFRmRDMAAlloc()

The resource manager handles the allocation and freeing of the four read DMA channels. This function allocates an RDMA channel, and returns the handle for the allocated channel.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMAAlloc(GF_HANDLE  RmHandle,
              GF_HANDLE *DmaHandle);
```

Parameters

RmHandle	Handle to the Rm allocated via call to GFRmOpen.
DmaHandle	Pointer to an allocated handle.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error. (Read DMA already released.)

GFRmRDMARelease()

The resource manager handles the allocation and freeing of the four read DMA channels. This function frees an allocated RDMA channel.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMARelease(GF_HANDLE  RmHandle,
                GF_HANDLE *DmaHandle);
```

Parameters

RmHandle	Handle to the Rm allocated via call to GFRmOpen.
DmaHandle	Pointer to the DMA handle to be released..

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmRDMASetupNONRect()

This function sets up the non-rectangular read.

- When the GF_RDMA_FLAGS_STR_HEADER flag is set, the buffer header is written to the memory by the RDMA client.

  In this case, when the CPU reads the data, the buffer header comes first and then the data. The buffer header contains information about the buffer– such as size, channel, etc.

  The following is an example of a typical sequence:

  i. Call setup with GF_RDMA_FLAGS_STR_HEADER flag set.

  ii. Read header with the GFRmRDMAReadHeader().

  iii. Read buffer size of data with GFRmRDMARead().

  The buffer size is in the buffer header.

- If the GF_RDMA_FLAGS_STR_HEADER flag is not set, there is no buffer header.

  In this case, it is assumed that the the size of the buffer is prefixed and known to the module API writers.

  The following is an example of a typical sequence:

  i. Call setup with GF_RDMA_FLAGS_STR_HEADER flag not set.

  ii. Read buffer size of data with GFRmRDMARead() function.

  The buffer size is known to the callers.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMASetupNONRect(GF_HANDLE      DmaHandle,
                     pRDMA_NONRECT  pReq);
```

Parameters

DmaHandle	Handle returned by **"GFRmRDMAAlloc()'**
pReq	Populated RDMA rectangular structure See "RDMA_NONRECT" on page 589.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmRDMASetupRect()

This function sets up the rectangular read.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMASetupRect(GF_HANDLE  DmaHandle,
                  pRDMA_RECT pReq);
```

Parameters

DmaHandle	Handle returned by **GFRmRDMAAlloc()**
pReq	Populated RDMA req structure. See "RDMA_RECT" on page 588.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmRDMAFIFOAvailIn32Bits()

This function reads the RDMA FIFO status register and returns the available number of FIFO slots. Each slot is 32 bits.

Function Prototype

```
GF_FUNC
NvU32  GFRmRDMAFIFOAvailIn32Bits(GF_HANDLE DmaHandle);
```

Parameters

DmaHandle	Handle returned by **GFRmRDMAAlloc()**

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmRDMARectRead()

This function reads memory from SC15 internal/external memory to the system memory pointed by the **dstAddr**.

The rectangular attributes passed must be the same rectangular attributes used when the RDMA was setup.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMARectRead(GF_HANDLE  DmaHandle,
                 void      *dstAddr,
                 NvU32      width,
                 NvU32      height);
```

Parameters

DmaHandle	Handle returned by **GFRmRDMAAlloc()**
dstAddr	Aligned or non-aligned pointer to the system memory destination. (Aligned pointer results in faster reads.)
width	Width in bytes to read
height	Height of the rectangular region

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmRDMARead()

This function reads memory from SC15 internal/external memory to the system memory pointed by the **dstAddr**.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMARead(GF_HANDLE  DmaHandle,
             void      *dstAddr,
             NvU32      sizeInBytes);
```

Parameters

DmaHandle	Handle returned by **GFRmRDMAAlloc()**
dstAddr	Aligned or non-aligned destination pointer. (Aligned pointer results in faster reads.)
sizeInBytes	Number of bytes to read

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmRDMAReadHeader()

This function reads the buffer header. Once this info is read, APIs will know how much data to expect to read.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMAReadHeader(GF_HANDLE            DmaHandle,
                   pRDMA_BUFFER_HEADER  header);
```

Parameters

DmaHandle	Handle returned by **GFRmRDMAAlloc()**
header	Pointer to RDMA header structure. See "RDMA_BUFFER_HEADER" on page 588.

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmRDMAMemCopy()

This function performs a linear copy from embedded memory to system memory, then calls **`GFRmRDMASetupRect()`** and **`GFRmRDMARead()`**.

Because this function converts linear reads to rectangular reads supported by the hardware, there is some translation. Therefore, if you are interested in rectangular reads, use the above specified functions directly. Calling this function incurs slight additional overhead.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMAMemCopy(GF_HANDLE          RmHandle,
                void              *dstAddr,
                pRDMA_MEMCOPY_REQ  pReq);
```

Parameters

`RmHandle`	Handle to the Rm allocated via call to GFRmOpen.
`dstAddr`	Pointer to system memory pointer.
`pReq`	Pointer to the "RDMA_MEMCOPY_REQ" on page 590.

Return Values

`GF_SUCCESS`	If successful.
`GF_ERROR`	If error.

GFRmRDMACleanup()

This function cleans up the RDMA FIFO by reading out any extra DWORDs that GFRmRDMARead might not have read.

In general, this function is not needed.

Function Prototype

```
GF_FUNC GF_RETTYPE
GFRmRDMACleanup(GF_HANDLE DmaHandle);
```

Parameters

DmaHandle	Handle returned by **GFRmRDMAAlloc()**

Return Values

GF_SUCCESS	If successful.
GF_ERROR	If error.

GFRmRDMA Structures

This section describes the following GFRmRDMA data structures:

RDMA_BUFFER_HEADER

This structure specifies the RDMA buffer header format.

For non-rectangular(or linear) reads RDMA expects a header in the memory. **`GFRmRDMAReadHeader()`** returns the header in this format.

See "GFRmRDMAReadHeader()" on page 585.

RDMA_BUFFER_HEADER Structure

```
typedef struct _RDMA_BUF_HEADER {
    NvU32 bufferSize;
    NvU32 raiseVector;
    NvU32 channel;
    NvU8  raiseEnable;
    NvU8  frameStart;
    NvU8  frameEnd;
    NvU8  largeHdr;
    NvU32 extHeader;
} RDMA_BUFFER_HEADER, *pRDMA_BUFFER_HEADER;
```

RDMA_RECT

This structure specifies setup information for the RDMA.

RDMA_Rect Structure

```
typedef struct _RDMA_RECT {

    NvU32 baseOffset;
       \\ Physical addess of the buffer in the SC15 Memory view

    eGFRmRDMAClientID clid;
        \\ Client ID. See "eGFRmRDMAClientID Enum" on page 590.
```

RDMA_Rect Structure (continued)

```
    NvU32 buffers;
    NvU32 stride;
    NvU32 width;
    NvU32 lines;
      // buffers/stride/width/lines of the rectangular buffer.
       These attributes must comply with the following hardware
       constraints:
       - If width is not a multiple of 4 bytes (one word)l
         then:
          - If stride equals width, stride should be the exact
            stride in bytes.
            (eg. line_stride[1:0] not equal to 0)
          - If stride is bigger than width, stride should be
            rounded down to the nearest word.
            (eg. line_stride[1:0] should be zero)

       For example:
          If width = 42 bytes, and stride = 82
             -> stride should be programmed to 80
          If width = 42 bytes, and stride = 42
             -> stride should be programmed to 42

    NvU32 flags;
       // Use the RDMA flags defined in
          "GF_RDMA Flags" on page 591.
} RDMA_RECT, *pRDMA_RECT;
```

RDMA_NONRECT

Setup structure for non-rectangular reads.

RDMA_NONRECT Structure

```
typedef struct _RDMA_NONRECT {

    NvU32 baseOffset;
      // Physical addess of the buffer in the SC15 Memory view *

    eGFRmRDMAClientID clid;
     // Client ID. See "eGFRmRDMAClientID Enum" on page 590.

    NvU32 buffers;
    NvU32 stride;
```

RDMA_NONRECT Structure

```
    NvU32 flags;
      // Use the RDMA flags defined in
         "GF_RDMA Flags" on page 591.
} RDMA_NONRECT, *pRDMA_NONRECT;
```

RDMA_MEMCOPY_REQ

Setup structure for memory copy.

RDMA_MEMCOPY_REQ Structure

```
typedef struct _RDMA_MEMCOPY_REQ {
    NvU32 baseOffset;
    NvU32 size;
    NvU32 flags;
} RDMA_MEMCOPY_REQ, *pRDMA_MEMCOPY_REQ;
```

GFRmRDMA Enumerations

eGFRmRDMAClientID Enum

Enumeration of client id's. SC15 has four RDMA (Read DMA) FIFOs. Modules specified below can be attached to the RDMA FIFOs.

```
typedef enum
{
    RDMA_CLID_CPU = 0,
    RDMA_CLID_DSP,
    RDMA_CLID_I2S,
    RDMA_CLID_SD,
    RDMA_CLID_MPEGE,
    RDMA_CLID_JPEGE,
    RDMA_CLID_EPP,
    RDMA_CLID_VI,
    RDMA_CLIDS
} eGFRmRDMAClientID;
```

GFRmRDMA Flags

GF_RDMA Flags

```
No swap option
#define GF_RDMA_FLAGS_SWAP_NONE             0x00000000

Swap bytes in a word. Example: 0xaabbccdd to 0xbbaaddcc
#define GF_RDMA_FLAGS_SWAP_BYTE_IN_WORD     0x00000001

Swap bytes in dword. Example: 0xaabbccdd to 0xddccbbaa
#define GF_RDMA_FLAGS_SWAP_BYTE_IN_DWORD    0x00000002

Swap word in dword. Example: 0xaabbccdd to 0xccddaabb
#define GF_RDMA_FLAGS_SWAP_WORD_IN_DWORD    0x00000003

#define GF_RDMA_FLAGS_SWAP_MASK             0x00000003

This flag is relevant only in non-rectangular reads. It tells
the RDMA engine to output the header so that the API sees a
header in the data stream.
#define GF_RDMA_FLAGS_STR_HEADER            0x00010000
```

Glossary of Technical Terms

A glossary explanation for an entry that is an abbreviation, such as an acronym, is generally limited to the unabbreviated form of the entry.

For additional entries related to the GFDxAPI, see "GFDxAPI Abbreviations and Acronyms" on page 136.

B

BIU. Bus interface unit.

G

GF. GoForce.

GFDxAPI. GoForce display and initialization API. See "Display API (GFDxAPI)" on page 123.

GFFDevAPI. GoForce file device API. See "File Device API (GFFDevAPI)" on page 523.

GFGxAPI. GoForce graphics API. See "Graphics API (GFGxAPI)" on page 153.

GFJxDecAPI. GoForce JPEG decoder API. See "JPEG Decoder API (GFJxDecAPI)" on page 399.

GFJxEncAPI. GoForce JPEG encoder API. See "JPEG Encoder API (GFJxEncAPI)" on page 375.

GFMxDecAPI. GoForce MPEG-4 decoder API. See "Low-Level MPEG-4 Decoder API (GFMxDecAPI)" on page 447.

GFMxEncAPI. GoForce MPEG-4 encoder API. See "MPEG-4 Encoder API (GFMxEncAPI)" on page 415.

GFRm. GoForce resource manager services. See "Resource Manager Services (GFRm)" on page 17.

GFSDK. GoForce software development kit. See "GoForce Software Development Kit (GFSDK)" on page 1.

GFVxAPI. GoForce video API. See "Video API (GFVxAPI)" on page 219.

GPIO. General purpose I/O.

H

Helper services. Services designed to isolate those library functions that are specific to a given programming language. See "Helper Services" on page 58.

L

LCD. Liquid crystal diode.

M

MIU. Memory interface unit.

O

ODM. Original design manufacturer.

OEM. Original equipment manufacturer.

P

PLL. Phase-locked loop.

PWM. Pulse width modulation.

R

ROP. Raster operation. See "Raster Operation (ROP) Codes" on page 214.

ROSC. Relaxation oscillator.

S

SRAM. Static random access memory.

STN. Supertwist nematic, a type of liquid crystal display.

Surface. Describes a two-dimensional memory region used by different GFSDK components. See "Surface Manager Services" on page 31.