

6 API REFERENCE

In This Chapter

The VisualDSP++ Kernel Application Programming Interface is a library of functions and macros that may be called from your application programs. Application programs depend on API functions to perform services that are basic to the VDK. These services include interrupt handling, scheduler management, thread management, semaphore management, events and event bits, and device drivers.

All of the VDK functions are written in the C++ programming language. You can use the object files of the API library in DSP systems based on ADSP-219x, ADSP-21xxx, and ADSP-TSxxx architectures.

This chapter describes the current release of the API library. Future releases may include additional functions.

This chapter provides information on the following topics:

- [“Calling Library Functions” on page 6-2](#)
- [“Linking Library Functions” on page 6-2](#)
- [“Working With VDK Library Header” on page 6-2](#)
- [“Passing Function Parameters” on page 6-3](#)

For reference information about the VDK library functions, see [page 6-41](#).

Calling Library Functions

To use an API function or macro, call it by name and provide the appropriate arguments. The name and arguments for each library entity appear on its reference page. Note that the function names are C and C++ function names. If you call a C run-time library function from an assembly language program, use its assembly version: prefix the function name with an underscore.

Similar to other functions, library functions should be declared. Declarations are supplied in the `vdk.h` header file. For more information about the kernel header file, see [page 6-2](#).

The reference pages appear in the “[Documented Library Functions](#)” section beginning on [page 6-41](#).

Linking Library Functions

When your code calls an API function, the call creates a reference resolved by the linker when linking your program. One way to direct the linker to the library’s location is to use the default VDK Linker Description File (VDK-*<your_target>.ldf*). The default VDK Linker Description File automatically direct the linker to the **.dlb* file in the `\lib` subdirectory of your VisualDSP++ installation.

If you do not use the default VDK LDF file, add the library file to your project’s LDF. Alternatively, use the compiler’s `-l` (library directory) switch to specify the library to be added to the link line. Library functions are not linked in the `.dxe` unless they are called.

Working With VDK Library Header

If your program calls an API library function, you include the `vdk.h` header file with the `#include` preprocessor command. The header file provide prototypes for all VDK public functions. The compiler uses

prototypes to check that each function is called with the correct arguments. The `vdk.h` file also provides declarations for user-accessible global variables, macros, type definitions, and enumerations.

Passing Function Parameters

All parameters passed through the VDK library functions described in [“Documented Library Functions” on page 6-5](#) are either passed by value or as constant objects. This means that the VDK does not modify any of the variables passed.

Library Naming Conventions

[Table 6-1](#) shows coding-style conventions that apply to the entities in the reference section.

Table 6-1. Library Notation Conventions

Notation	Description
<code>inSemaphoreID</code>	Variable parameters are written with the first letter lowercase and uppercase otherwise.
<code>VDK_Ticks</code>	VDK-defined types are written in uppercase.
<code>VDK_SET_EVENTBIT_</code>	Preprocessor macros are written in uppercase with words separated by underscores and a trailing underscore.
<code>kPriority1</code>	Constants are written in uppercase and prefixed with a “k”.

By following the function naming convention described in [Table 6-2](#), you can review VDK sources or documentation and can recognize whether the non-keyword is a variable, parameter, macro, or a constant.

Table 6-2. Function Naming Convention

Notation	Description
<code>VDK_</code>	C applications use callable function names prefixed by “VDK_” to clearly distinguish VDK library functions from user functions.
<code>VDK::</code>	C++ callable functions are located in the “VDK” namespace, thus function names are preceded by “VDK::”.
<code>VDK_Yield(void)</code>	The remaining portion of the function name is written in uppercase.

Documented Library Functions

[Table 6-3](#) through [Table 6-13](#) list the VDK library entities included in the current software release. These tables list the library entities grouped by the service the grouping provides. The reference sections beginning on [page 6-41](#) appear in the alphabetic order.

Table 6-3. Interrupt Handling Functions

Function Name	Reference Page
PopCriticalRegion()	page 6-41
PopNestedCriticalRegions()	page 6-42
PushCriticalRegion()	page 6-47

Table 6-4. Scheduler Management Functions

PopNestedUnscheduledRegions()	page 6-43
PopUnscheduledRegion()	page 6-44
PushUnscheduledRegion()	page 6-48

Table 6-5. Thread and System Information Functions

GetThreadHandle()	page 6-25
GetThreadID()	page 6-26
GetThreadStatus()	page 6-27
GetThreadType()	page 6-28

Documented Library Functions

Table 6-5. Thread and System Information Functions (Cont'd)

GetUptime()	page 6-29
GetVersion()	page 6-30
LogHistoryEvent()	page 6-31

Table 6-6. Thread Creation and Destruction Functions

CreateThread()	page 6-13
DestroyThread()	page 6-14
FreeDestroyedThreads()	page 6-17

Table 6-7. Thread Error Management Functions

ClearThreadError()	page 6-11
DispatchThreadError()	page 6-16
GetLastThreadError()	page 6-21
GetLastThreadErrorValue()	page 6-22
SetThreadError()	page 6-55

Table 6-8. Thread Priority Management Functions

GetPriority()	page 6-23
ResetPriority()	page 6-51
SetPriority()	page 6-54

Table 6-9. Thread Scheduling Control Functions

Sleep()	page 6-52
Yield()	page 6-58

Table 6-10. Semaphore Management Functions

GetSemaphoreValue()	page 6-24
MakePeriodic()	page 6-33
PendSemaphore()	page 6-39
PostSemaphore()	page 6-46
SetEventBit()	page 6-53

Table 6-11. Event and EventBit Functions

ClearEventBit()	page 6-10
GetEventBitValue()	page 6-18
GetEventData()	page 6-19
GetEventValue()	page 6-20
LoadEvent()	page 6-32
PendEvent()	page 6-37
SetEventBit()	page 6-53

Documented Library Functions

Table 6-12. Device Driver Functions

CloseDevice()	page 6-12
OpenDevice()	page 6-35
OpenDevice()	page 6-35
PendDeviceFlag()	page 6-36
SyncRead()	page 6-56
SyncRead()	page 6-56
SyncWrite()	page 6-57

Table 6-13. Assembly Macros

Macro Name	Reference Page
VDK_ISR_ACTIVATE_DEVICE_DRIVER_()	page 6-60
VDK_ISR_CLEAR_EVENTBIT_()	page 6-61
VDK_ISR_LOG_HISTORY_EVENT_()	page 6-62
VDK_ISR_POST_SEMAPHORE_()	page 6-63
VDK_ISR_SET_EVENTBIT_()	page 6-64

Description Format

The following format applies to all of the entries in the library reference section.

C Prototype

provides C prototype (as it found in `vdk.h`) describing the interface to the function

C++ Prototype

provides C++ prototype (as it found in `vdk.h`) describing the interface to the function

Description

describes the function's operation

Parameters

describes the function's parameters

Scheduling

adds a note when the function invokes the scheduler

Determinism

specifies whether the function is deterministic

Errors Thrown

specifies error-handling routines

ClearEventBit

C Prototype

```
void VDK_ClearEventBit(EventBitID inEventBitID);
```

C++ Prototype

```
void VDK::ClearEventBit(EventBitID inEventBitID);
```

Description

This function clears the value of the EventBit and sets it to FALSE, NULL, or 0. Once the EventBit has been cleared, the value of each of the dependent events is recalculated.

Parameters

`inEventBitID` is the system EventBit to clear.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

Not Deterministic.

Errors Thrown

`kUnknownEventBit`

ClearThreadError

C Prototype

```
void VDK_ClearThreadError (void);
```

C++ Prototype

```
void VDK::ClearThreadError(void);
```

Description

This function sets the running thread's error status to `kNoError` and the error value to 0.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

CloseDevice

C Prototype

```
void VDK_CloseDevice(VDK_DeviceDescriptor inDD);
```

C++ Prototype

```
void VDK::CloseDevice(VDK::DeviceDescriptor inDD);
```

Description

This function closes the specified device. The function calls the device dispatch function of the device that has been opened with `inDD`.

Parameters

`inDD` is the Device Descriptor returned from the `OpenDevice`.

Scheduling

This function does not invoke the scheduler, but the user-written Device Driver can call the scheduler.

Determinism

Constant Time. Note that this function calls user-written device driver code that may not be deterministic.

Errors Thrown

`kDD_BadDeviceDescriptor`

CreateThread

C Prototype

```
VDK_ThreadID VDK_CreateThread(ThreadType inType);
```

C++ Prototype

```
VDK::ThreadID VDK::CreateThread(ThreadType inType);
```

Description

This function creates a thread of the specified type and returns the new thread.

Parameters

`inType` corresponds to a `Thread Type` defined in the `Vdk.h` and `Vdk.cpp` files. These files contain the default values for the stack size, initial priority, and other properties.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

Not Deterministic.

Errors Thrown

`kUnknownThreadType`, `kThreadCreationFailure`

DestroyThread

C Prototype

```
void VDK_DestroyThread(VDK_ThreadID  inThreadID,  
                       bool           inDestroyNow);
```

C++ Prototype

```
void VDK::DestroyThread(VDK::ThreadID inThreadID,  
                        bool           inDestroyNow);
```

Description

This function initiates the process of removing the specified thread from the system. Although the scheduler never runs the thread again once this function completes, the kernel deallocates the memory resources associated with the thread using a low-priority level thread (IDLE). Any references to the destroyed thread are invalid and may throw an error.

Parameters

`inThreadID` specifies the thread to remove from the system.

`inDestroyNow` indicates whether the thread's memory is to be recovered now (`true`) or in the low-priority IDLE thread (`false`).

Scheduling

This function invokes the scheduler and results in a context switch only if a thread passes itself to `DestroyThread()`.

Determinism

Constant Time.

Errors Thrown

`kUnknownThread`

DeviceIOctl

C Prototype

```
int VDK_DeviceIOctl (VDK_DeviceDescriptor inDD,
                    void                *inCommand,
                    char                *inParameters);
```

C++ Prototype

```
int VDK::DeviceIOctl (VDK::DeviceDescriptor inDD,
                    void                *inCommand,
                    char                *inParameters);
```

Description

This function controls the specified device. The Command and Parameters are passed unchanged to the Device Driver.

Parameters

`inDD` is the Device Descriptor returned from the `OpenDevice`.

`inCommand` are the Device Driver's Specific Commands.

`inParameters` are the Device Driver's specific parameters for the above commands.

Scheduling

This function does not invoke the scheduler, but the user-written Device Driver can call the scheduler.

Determinism

Constant Time. Note that this function calls user-written device driver code that may not be deterministic.

Errors Thrown

`kDD_BadDeviceDescriptor`

DispatchThreadError

C Prototype

```
int VDK_DispatchThreadError (VDK_SystemError inErr,  
                             const int      inVal);
```

C++ Prototype

```
int VDK::DispatchThreadError(VDK::SystemError inErr,  
                             const int      inVal);
```

Description

This function sets the error and error's value in the currently running thread; also calls its error function.

Parameters

`inErr` is the error enumeration. See [“SystemError” on page 5-16](#) for more information about errors.

`inVal` is the value whose meaning determined by the error enumeration.

Scheduling

This function invokes the scheduler.

Determinism

Not Deterministic.

Errors Thrown

None.

FreeDestroyedThreads

C Prototype

```
void VDK_FreeDestroyedThreads(void);
```

C++ Prototype

```
void VDK::FreeDestroyedThreads(void);
```

Description

This function frees the memory held by the destroyed threads whose resources have not been released by the `IDLE` thread.

Parameters

None

Scheduling

This function does not invoke the scheduler.

Determinism

Not Deterministic.

Errors Thrown

None.

GetEventBitValue

C Prototype

```
bool VDK_GetEventBitValue(EventBitID inEventBitID);
```

C++ Prototype

```
bool VDK::GetEventBitValue(EventBitID inEventBitID);
```

Description

This function returns the value of the EventBit with the given ID.

Parameters

`inEventBitID` is the system EventBit to query.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

`kUnknownEventBit`

GetEventData

C Prototype

```
VDK_EventData VDK_GetEventData(EventID inEventID);
```

C++ Prototype

```
VDK::EventData VDK::GetEventData(EventID inEventID);
```

Description

This function returns the Mask, EventBit vector, and logic associated with the queried event. Threads can use this function to get an event's current values.

Parameters

`inEventID` is the event to query.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

`kUnknownEvent`

GetEventValue

C Prototype

```
bool VDK_GetEventValue(EventID inEventID);
```

C++ Prototype

```
bool VDK::GetEventValue(EventID inEventID);
```

Description

This function returns the value of the event with the given ID.

Parameters

`inEventID` is the event to be queried.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

`kUnknownEvent`

GetLastThreadError

C Prototype

```
VDK_SystemError VDK_GetLastThreadError (void);
```

C++ Prototype

```
VDK::SystemError VDK::GetLastThreadError(void);
```

Description

This function returns the running thread's most recent error. See [“SystemError” on page 5-16](#) for more information about errors.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

GetLastThreadErrorValue

C Prototype

```
int VDK_GetLastThreadErrorValue (void);
```

C++ Prototype

```
int VDK::GetLastThreadErrorValue(void);
```

Description

This function returns the value parameter of the most recent error thrown by the running thread. See [“SystemError” on page 5-16](#) for more information about errors.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

GetPriority

C Prototype

```
VDK_Priority VDK_GetPriority(const ThreadID inThreadID);
```

C++ Prototype

```
VDK::Priority VDK::GetPriority(const ThreadID inThreadID);
```

Description

This function returns the priority of the named thread.

Parameters

`inThreadID` is the thread whose priority is being queried.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

`kUnknownThread`

GetSemaphoreValue

C Prototype

```
bool VDK_GetSemaphoreValue(SemaphoreID inSemaphoreID);
```

C++ Prototype

```
bool VDK::GetSemaphoreValue(SemaphoreID inSemaphoreID);
```

Description

This function returns the value of the semaphore with the given ID.

Parameters

`inSemaphoreID` is the semaphore to quire.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

`kUnknownSemaphore`

GetThreadHandle

C Prototype

```
void** VDK_GetThreadHandle (void);
```

C++ Prototype

```
void** VDK::GetThreadHandle(void);
```

Description

This function returns a pointer to a thread's user-defined, allocated data pointer.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

GetThreadID

C Prototype

```
ThreadID VDK_GetThreadID (void);
```

C++ Prototype

```
ThreadID VDK::GetThreadID (void);
```

Description

This function returns the ID of the currently running thread.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

GetThreadStatus

C Prototype

```
VDK_ThreadStatus VDK_GetThreadStatus(const ThreadID inThreadID);
```

C++ Prototype

```
VDK::ThreadStatus VDK::GetThreadStatus(const VDK::ThreadID inThreadID);
```

Description

This function reports the enumerated status of the thread.

Parameters

`inThreadID` is the thread whose status is being queried.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

GetThreadType

C Prototype

```
VDK_ThreadType VDK_GetThreadType(const ThreadID inThreadID);
```

C++ Prototype

```
VDK::ThreadType VDK::GetThreadType(const ThreadID inThreadID);
```

Description

This function returns the Thread Type used to create the thread. The Thread Type is defined in the `Vdk.h` and `Vdk.cpp` files. For more information about the Thread Types, refer to [“Thread Types” on page 4-2](#).

Parameters

`inThreadID` is the thread to query.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

`kUnknowThread`

GetUptime

C Prototype

```
VDK_Ticks VDK_GetUptime(void);
```

C++ Prototype

```
VDK::Ticks VDK::GetUptime(void);
```

Description

This function returns the time in ticks since the last system reset.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

GetVersion

C Prototype

```
VDK_VersionStruct VDK_GetVersion(void);
```

C++ Prototype

```
VDK::VersionStruct VDK::GetVersion(void);
```

Description

This function returns a standard VDK `VersionStruct`. For information about the `VersionStruct`, see [page 5-18](#).

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

LogHistoryEvent

C Prototype

```
VDK_LogHistoryEvent(VDK_HistoryID inEnum, int inValue);
```

C++ Prototype

```
VDK::LogHistoryEvent(VDK::HistoryID inEnum, int inValue);
```

Description

This function adds a record to the history buffer. The function is null if your project is not linked with the instrumented libraries.

Parameters

`inEnum` is the enumeration value for this type of event.

`inValue` is the value defined by Enumeration.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

LoadEvent

C Prototype

```
void VDK_LoadEvent(EventID inEventID,  
                   const VDK_EventData inEventData);
```

C++ Prototype

```
void VDK::LoadEvent(EventID inEventID,  
                    const VDK::EventData inEventData);
```

Description

This function loads the event's data (Mask, EventBits vector, and logic) associated with the event.

Parameters

`inEventID` is the event to be re-initialized.

`inEventData` contains the new values for the event.

Scheduling

This function causes the value of the event to be recalculated; invokes the scheduler; and may result in a context switch.

Determinism

Not Deterministic.

Errors Thrown

`kUnknownEvent`

MakePeriodic

C Prototype

```
void VDK_MakePeriodic(SemaphoreID  inSemaphoreID,
                     VDK_Ticks    inDelay,
                     VDK_Ticks    inPeriod);
```

C++ Prototype

```
void VDK::MakePeriodic(SemaphoreID inSemaphoreID,
                      VDK_Ticks    inDelay,
                      VDK_Ticks    inPeriod);
```

Description

This function directs the scheduler to post the specified semaphore after `inDelay` number of ticks. Thereafter, every `inPeriod` ticks, the semaphore is posted and the scheduler is invoked. This allows the running thread to acquire the signal and, if the thread is at the highest priority level, to continue execution.

To be periodic, the running thread must repeat in sequence: performing a task and then pending on the semaphore passed in (thus blocking). Note that this differs from sleeping at the completion of the task— the periodicity does not depend on the thread duration or time allocated to other threads.

Parameters

`inSemaphoreID` is the semaphore to block on.

`inDelay` is the number of ticks before the first posting of the semaphore. `inDelay` must be ≥ 1 .

`inPeriod` is the number of ticks to sleep at each cycle after the first cycle. `inPeriod` must be ≥ 1 .

Documented Library Functions

Scheduling

This function does not invoke the scheduler.

Determinism

Not Deterministic.

Errors Thrown

`kUnknownSemaphore`, `kInvalidPeriod`, `kInvalidDelay`,
`kAlreadyPeriodic`

OpenDevice

C Prototype

```
VDK_DeviceDescriptor VDK_OpenDevice (DeviceID inIDNum,
                                     char      *inFlags);
```

C++ Prototype

```
VDK::DeviceDescriptor VDK::OpenDevice(DeviceID inIDNum,
                                     char      *inFlags);
```

Description

This function opens the specified device.

Parameters

`inIDNum` is the Device ID number.

`inFlags` are the Flags to be passed to Device Driver “OPEN”.

Scheduling

This function does not call the scheduler, but the user-written Device Driver can call the scheduler.

Determinism

Constant Time. Note that this function calls user-written device driver code that may not be deterministic.

Errors Thrown

`kDD_BadDeviceID`, `kDD_OpenFailure`

PendDeviceFlag

C Prototype

```
void VDK_PendDeviceFlag (DeviceFlagID    inFlagID,  
                        VDK_Ticks        inTimeout);
```

C++ Prototype

```
void VDK::PendDeviceFlag(DeviceFlagID    inFlagID,  
                        VDK::Ticks        inTimeout);
```

Description

This function allows a thread to block on the specified DeviceFlag. The thread is blocked and swapped out. Once the DeviceFlag is made available via [PostDeviceFlag\(\)](#), *all* threads waiting for this flag are made ready to run.

Parameters

`inFlagID` is the Device Descriptor returned from the [OpenDevice](#).

`inTimeout` is the number of ticks to wait before timeout occurs.

Scheduling

This function invokes the scheduler.

Determinism

Not Deterministic.

Errors Thrown

`kBlockInInvalidRegion`, `kDD_InvalidDeviceFlag`, `kDD_DeviceTimeout`

PendEvent

C Prototype

```
void VDK_PendEvent(EventID inEventID, VDK_Ticks inTimeout);
```

C++ Prototype

```
void VDK::PendEvent (EventID inEventID, VDK::Ticks inTimeout);
```

Description

This function provides the mechanism allowing threads to pend on events. If the named event calculates as available, execution returns to the running thread. If the event is *not* available, the thread pauses execution until the event is available. When the event becomes available, *all* threads pending on the event are moved to the ready queue.

If the thread does not resume execution within *timeout* number of clock ticks, the thread's re-entry point is changed to its error function. If the value of *timeout* is passed as *zero*, then the thread may pend indefinitely.

Parameters

inEventID is the event on which the thread pends.

inTimeout specifies the duration in ticks for which the thread pends on the event.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

Available Event: Constant Time.

Unavailable Event with timeout: Not Deterministic.

Unavailable Signal without timeout: Constant time plus a context switch.

Documented Library Functions

Errors Thrown

`kEventTimeout`, `kUnknownEvent`, `kBlockInInvalidRegion`.

Debug Only: `kDbgPossibleBlockInRegion`

PendSemaphore

C Prototype

```
void VDK_PendSemaphore(SemaphoreID    inSemaphoreID,
                       VDK_Ticks      timeout);
```

C++ Prototype

```
void VDK::PendSemaphore(SemaphoreID    inSemaphore,
                        VDK::Ticks      timeout);
```

Description

This function provides the mechanism allowing threads to pend on semaphores. If the named semaphore is available, the semaphore becomes unavailable and processor control returns to the running thread. If the semaphore is *not* available, the thread pauses execution until the semaphore is posted. If the thread does not resume execution within `timeout` number of clock ticks, the thread's re-entry point is changed to its error function. If the value of `timeout` is passed as zero, then the thread may pend indefinitely.

Parameters

`inSemaphoreID` is the semaphore on which the thread pends.

`timeout` specifies the duration in ticks for which the thread pends on the semaphore.

Scheduling

This function invokes the scheduler and may result in a context switch.

Documented Library Functions

Determinism

Available semaphore: Constant Time.

Unavailable semaphore with timeout: Not Deterministic.

Unavailable semaphore without timeout: Constant time plus a context switch.

Errors Thrown

`kSemaphoreTimeout`, `kUnknownSemaphore`, `kBlockInInvalidRegion`.

Debug Only: `kDbgPossibleBlockInRegion`.

PopCriticalRegion

C Prototype

```
void VDK_PopCriticalRegion(void);
```

C++ Prototype

```
void VDK::PopCriticalRegion(void);
```

Description

This function re-enables interrupts following a sequence of automatically executed instructions (a critical region). Note that critical regions may be nested. A count is maintained to ensure that each entered critical region calls `PopCriticalRegion()` before interrupts are restored. The kernel ignores additional calls to `PopCriticalRegion()` while interrupts are enabled.

This function does not change the interrupt mask.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

Debug only: `kDbgPopUnderflow`

PopNestedCriticalRegions

C Prototype

```
void VDK_PopNestedCriticalRegions(void);
```

C++ Prototype

```
void VDK::PopNestedCriticalRegions(void);
```

Description

This function re-enables interrupts following a sequence of automatically executed instructions (a critical region). This function ignores and resets the current count of nested protected regions. The kernel ignores additional calls to this function while interrupts are enabled.

This function does not change the interrupt mask.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

Debug only: `kDbgPopUnderflow`

PopNestedUnscheduledRegions

C Prototype

```
void VDK_PopNestedUnscheduledRegions(void);
```

C++ Prototype

```
void VDK::PopNestedUnscheduledRegions(void);
```

Description

This function restores scheduling operations in the priority-based pre-emptive mode. This function ignores and resets the current count of nested unscheduled regions. While scheduling is enabled, the kernel ignores additional calls to this function.

Parameters

None.

Scheduling

This function invokes the scheduler and may result in a context switch. If scheduling is already enabled, this function does nothing and a context switch does not occur.

Determinism

Constant Time.

Errors Thrown

Debug only: `kDbgPopUnderflow`

PopUnscheduledRegion

C Prototype

```
void VDK_PopUnscheduledRegion(void);
```

C++ Prototype

```
void VDK::PopUnscheduledRegion(void);
```

Description

This function restores scheduling operations in the priority-based pre-emptive mode. Use function as a close bracket call to [PushUnscheduledRegion\(\)](#). Note that unscheduled regions may be nested. A nesting count is maintained to ensure that each entered unscheduled region calls [PopUnscheduledRegion\(\)](#) before scheduling is resumed.

The kernel ignores additional calls to this function while scheduling is enabled.

Parameters

None.

Scheduling

This function can invoke the scheduler and result in a context switch. If scheduling is already enabled, this function does nothing, and a context switch does not occur.

Determinism

Constant Time.

Errors Thrown

Debug only: `kDbgPopUnderflow`

PostDeviceFlag

C Prototype

```
void VDK_PostDeviceFlag (DeviceFlagID inFlagID);
```

C++ Prototype

```
void VDK::PostDeviceFlag(DeviceFlagID inFlagID);
```

Description

This function posts the specified DeviceFlag. Once the DeviceFlag is made available, *all* threads waiting for the flag are in the ready to run state.

Parameters

`inFlagID` is the Device Descriptor returned from the OpenDevice.

Scheduling

This function invokes the scheduler.

Determinism

Not Deterministic.

Errors Thrown

None.

PostSemaphore

C Prototype

```
void VDK_PostSemaphore(SemaphoreID inSemaphoreID);
```

C++ Prototype

```
void VDK::PostSemaphore(SemaphoreID inSemaphoreID);
```

Description

This function provides the mechanism by which threads post semaphores. Note that Interrupt Service Routines must use a different interface, as described in [“VDK_ISR_POST_SEMAPHORE_” on page 6-63](#).

Parameters

`inSemaphoreID` is the semaphore to post.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

No thread pending: Constant time TBD.

Low priority thread pending: Constant time TBD.

High priority thread pending: Constant time TBD plus a context switch.

Errors Thrown

`kUnknownSemaphore`

PushCriticalRegion

C Prototype

```
void VDK_PushCriticalRegion(void);
```

C++ Prototype

```
void VDK::PushCriticalRegion(void);
```

Description

This function disables interrupts to enable automatic execution of a protected region of code. Note that protected regions may be nested. A count is maintained to ensure that an equal number of calls to `PopCriticalRegion()` has been made before restoring interrupts.

This function does not change the interrupt mask.

Parameters

None.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

PushUnscheduledRegion

C Prototype

```
void VDK_PushUnscheduledRegion(void);
```

C++ Prototype

```
void VDK::PushUnscheduledRegion(void);
```

Description

This function disables the scheduler. Subsequent calls to the `pend`, `post`, `SetEventBit()`, and `ClearEventBit()` functions allow to operate on signals as normal. The difference is that the high-priority thread is not re-selected from the ready queue. This permits the currently running thread to continue execution without being pre-empted by another thread while the interrupts are enabled. If scheduling is already disabled, the kernel ignores the `PushUnscheduledRegion()` call.

A count of nested unscheduled regions is maintained. Each entered unscheduled region must be matched with a call to `PopUnscheduledRegion()` before scheduling is resumed.

Scheduling

Note that scheduling may be re-enabled after a call to `PushUnscheduledRegion()` by the scheduler itself to prevent a deadlock. When scheduling is disabled and a thread calls the `pend` function to access an unavailable signal, the following occurs:

- the thread pends (blocks) on a signal passed in
- the scheduler resumes automatically
- a context switch occurs, allowing the highest priority ready thread to execute

Parameters

None.

Determinism

Constant Time.

Errors Thrown

None.

RemovePeriodic

C Prototype

```
void VDK_RemovePeriodic(SemaphoreID inSemaphoreID);
```

C++ Prototype

```
void VDK::RemovePeriodic(SemaphoreID inSemaphoreID);
```

Description

This function stops posting the specified semaphore periodically. Trying to stop a non-periodic semaphore has no effect and raises a run-time error.

Parameters

`inSemaphoreID` is the semaphore which posting to be non-periodic.

Scheduling

This function does not invoke the scheduler.

Determinism

Not Deterministic.

Errors Thrown

`kUnknownSemaphore`, `kNonperiodicSemaphore`

ResetPriority

C Prototype

```
void VDK_ResetPriority(const ThreadID inThreadID);
```

C++ Prototype

```
void VDK::ResetPriority(const ThreadID inThreadID);
```

Description

This function restores the priority of the named thread to the default value specified in the thread's template.

Parameters

`inThreadID` is the thread whose priority to reset.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

Not Deterministic.

Errors Thrown

`kUnknownThread`

Sleep

C Prototype

```
void VDK_Sleep(VDK_Ticks inSleepTicks);
```

C++ Prototype

```
void VDK::Sleep(VDK::Ticks inSleepTicks);
```

Description

This function causes a thread to pause execution for at least the given number of clock ticks. Once delay ticks have elapsed, the calling thread is in the ready-to-run state. The thread resumes execution only if it is the highest priority thread with ready status. The minimum delay is 1.

Parameters

`inSleepTicks` is the duration in ticks for which the thread should sleep.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

Not Deterministic.

Errors Thrown

`kInvalidDelay`

SetEventBit

C Prototype

```
void VDK_SetEventBit(EventBitID inEventBitID);
```

C++ Prototype

```
void VDK::SetEventBit(EventBitID inEventBitID);
```

Description

This function sets the value of the `eventBit`. Once the `eventBit` has been set (meaning that its value equals to `TRUE`, 1, occurred, etc.), the value of all dependent events is recalculated.

Parameters

`inEventBitID` is the system `eventBit` to set.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

Not Deterministic.

Errors Thrown

`kUnknownEventBit`

SetPriority

C Prototype

```
void VDK_SetPriority(const ThreadID      inThreadID,  
                   const VDK_Priority  inPriority);
```

C++ Prototype

```
void VDK::SetPriority(const ThreadID      inThreadID,  
                   const VDK::Priority  inPriority);
```

Description

This function dynamically sets the priority of the named thread overriding the default value. All threads are given an initial priority level at creation time. The thread template ID specifies the priority initial value.

Parameters

`inPriority` is the new priority level.

`inThreadID` is the thread to modify.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

Not Deterministic.

Errors Thrown

`kUnknownThread`, `kInvalidPriority`

SetThreadError

C Prototype

```
void VDK_SetThreadError (VDK_SystemError inErr, int inVal);
```

C++ Prototype

```
void VDK::SetThreadError(VDK::SystemError inErr, int inVal);
```

Description

This function sets the running thread's error value.

Parameters

`inErr` is the error enumeration. See [“SystemError” on page 5-16](#) for more information about errors.

`inVal` is the value whose meaning is determined by the error enumeration.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

SyncRead

C Prototype

```
unsigned int VDK_SyncRead(VDK_DeviceDescriptor inDD,  
                          char *outBuffer,  
                          const unsigned int inSize,  
                          VDK_Ticks inTimeout);
```

C++ Prototype

```
unsigned int VDK::SyncRead(VDK::DeviceDescriptor inDD,  
                           char *outBuffer,  
                           const unsigned int inSize,  
                           VDK::Ticks inTimeout);
```

Description

This function synchronously reads data from the specified device.

Parameters

`inDD` is the Device Descriptor returned from the `OpenDevice`.

`outBuffer` is the address of the data buffer to be filled by the device.

`inSize` is the number of words to be read from the device.

`inTimeout` is the number of ticks to wait before timeout occurs.

Scheduling

This function does not call the scheduler, but the user-written Device Driver can call the scheduler.

Determinism

Constant Time. Note that this function calls user-written device driver code that may not be deterministic.

Errors Thrown

`kDD_BadDeviceDescriptor`, `kDD_DeviceTimeout`

SyncWrite

C Prototype

```
unsigned int VDK_SyncWrite(VDK_DeviceDescriptor inDD,
                          char *outBuffer,
                          const unsigned int inSize,
                          VDK_Ticks inTimeout);
```

C++ Prototype

```
unsigned int VDK::SyncWrite(VDK::DeviceDescriptor inDD,
                            char *outBuffer,
                            const unsigned int inSize,
                            VDK::Ticks inTimeout);
```

Description

This function synchronously writes data to the specified device.

Parameters

`inDD` is the Device Descriptor returned from the `OpenDevice`.

`outBuffer` is the address of the data buffer to be read by the device.

`inSize` is the number of words to be written to the device.

`inTimeout` is the number of ticks to wait before timeout occurs.

Scheduling

This function does not call the scheduler, but the user-written Device Driver can call the scheduler.

Determinism

Constant Time. Note that this function calls user-written device driver code that may not be deterministic.

Errors Thrown

`kDD_BadDeviceDescriptor`, `kDD_DeviceTimeout`

Yield

C Prototype

```
void VDK_Yield(void);
```

C++ Prototype

```
void VDK::Yield(void);
```

Description

This function yields control of the processor. This function moves the thread to the end of the wait queue of threads at its priority level. If `Yield()` is called from a thread at a priority level using round-robin multithreading, the call also yields the remainder of the thread's time slice.

Parameters

None.

Scheduling

This function invokes the scheduler and may result in a context switch.

Determinism

Constant time and conditional context switch.

Errors Thrown

`kBlockInInvalidRegion`

Debug Only: `kDbgPossibleBlockInRegion`

Assembly Macros

This section describes the assembly language macros representing the Application Programming Interface between the kernel and Interrupt service Routines.

The timing of interrupts is controlled by the processor's hardware. The processor state upon entry likely does not comply with the C run-time model for the entry to a typical function. Therefore, the user must write Interrupt Service Routines entirely in assembly. Alternately, the user can coerce the system into compliance with the C run-time model by performing a full context switch. For efficiency, the second method is strongly recommended.

Even if the user chooses to force compliant with the C run-time model, ISRs cannot use the C language API described in the previous sections. With regard to signals, ISRs are limited to posting only. An ISR cannot pend on a signal because it would not make sense for it to be “blocked” if the signal is not available.

Provided assembly language macros represent the interface to the kernel from ISRs. Each macro saves and restores all registers it uses.

Note that implemented in the assembly language threads can also use the macros. They do not require the C run-time model to be intact.

VDK_ISR_ACTIVATE_DEVICE_DRIVER_

Prototype

```
VDK_ACTIVATE_DEVICE_DRIVER_(DeviceID inID);
```

Description

Execute Device Driver *inID* when execution returns to the thread domain.

Parameters

inID is the Device Driver to run.

Scheduling

This macro invokes the Device Driver when execution returns to the thread domain.

Determinism

Constant Time.

VDK_ISR_CLEAR_EVENTBIT_

Prototype

```
VDK_ISR_CLEAR_EVENTBIT_(EventBitID inEventBit)
```

Description

This macro clears the value of the `inEventBit` by setting it to `FALSE` or `0`. All eventBit clears, which occur in the interrupt scheduling domain, are processed on return to the thread scheduling domain.

Parameters

`inEventBit` is the system eventbit to clear.

Scheduling

If `inEventBit` had been set, this macro invokes the scheduler when execution returns to the thread domain. This allows the value of all dependent events to be recalculated.

Determinism

Constant Time.

VDK_ISR_LOG_HISTORY_EVENT_

Prototype

```
VDK_LOG_HISTORY_EVENT_(HistoryID    inEnum,  
                        int          inVal,  
                        ThreadID     inThreadID);
```

Description

This macro adds a record to the history buffer. The macro is NULL if the VDK_DEBUG_BUILD_ macro is undefined.

Parameters

`inEnum` is the enumeration value for this type of event.

`inVal` is the information defined by the enumeration.

`inThreadID` is the ThreadID to be stored with History Event.

Scheduling

This function does not invoke the scheduler.

Determinism

Constant Time.

Errors Thrown

None.

VDK_ISR_POST_SEMAPHORE_

Asm Prototype

```
VDK_POST_SEMAPHORE_(SemaphoreID inSemaphoreID)
```

Description

This macro posts the named semaphore. If the semaphore is already available, the macro has no effect. *All* semaphore post sets, which occur in the interrupt scheduling domain, are processed on return to the thread scheduling domain.

Parameters

`inSemaphoreID` is the semaphore to post.

Scheduling

If a thread is pended on `inSemaphoreID`, this macro invokes the scheduler when execution returns to the thread domain

Determinism

Constant Time.

VDK_ISR_SET_EVENTBIT_

Prototype

```
VDK_SET_EVENTBIT_(EventBitID inEventBit)
```

Description

This macro sets the value of `inEventBit` by setting it to `TRUE` or `1`. *All* `eventBit` sets, which occur in the interrupt scheduling domain, are processed on return to the thread scheduling domain.

Parameters

`inEventBit` the system `eventBit` to set.

Scheduling

If `inEventBit` has been cleared, this macro invokes the scheduler when execution returns to the thread domain. This allows the value of all dependent events to be recalculated.

Determinism

Constant Time.