

5 DATA TYPES

In This Chapter

Your VDK software comes with the predefined set of data types. This chapter describes the current release of the kernel. Future releases may include more types.

The VDK data types are summarized in [Table 5-1](#). A description of each type starts on [page 5-3](#).

Table 5-1. VDK Data Types

VDK Data Type	Reference Page
Bitfield	page 5-3
DeviceDescriptor	page 5-4
DeviceDispatchID	page 5-5
DeviceDispatchUnion	page 5-6
DeviceFlagID	page 5-8
EventBitID	page 5-9
EventID	page 5-9
EventData	page 5-10

Table 5-1. VDK Data Types (Cont'd)

VDK Data Type	Reference Page
Priority	page 5-11
SemaphoreID	page 5-12
ThreadID	page 5-12
ThreadStatus	page 5-13
ThreadType	page 5-14
Ticks	page 5-15
SystemError	page 5-16
VersionStruct	page 5-18

Bitfield

The Bitfield type is used to store the bit pattern. The size of a Bitfield item is the size of a data word:

- sixteen bits on ADSP-219x DSPs
- thirty-two bits on ADSP-21xxx and ADSP-TSxxx DSPs

In C:

```
typedef struct
{
    unsigned int      Bit0:1;
    ...
    unsigned int      BitN:1;
} VDK_Bitfield;
```

In C++:

```
typedef struct
{
    unsigned int      Bit0:1;
    ...
    unsigned int      BitN:1;
} VDK::Bitfield;
```

DeviceDescriptor

The DeviceDescriptor type is used to store the unique identifier of an Open Device. The value is obtained dynamically as the return value from VDK_OpenDevice().

In C:

```
typedef unsigned int VDK_DeviceDescriptor;
```

In C++:

```
typedef unsigned int VDK::DeviceDescriptor;
```

DeviceDispatchID

The DeviceDispatchID type enumerates Device Driver Dispatch commands.

In C:

```
enum VDK_DeviceDispatchID
{
    VDK_kDD_Init,
    VDK_kDD_Activate,
    VDK_kDD_Open,
    VDK_kDD_Close,
    VDK_kDDSyncRead,
    VDK_kDDSyncWrite,
    VDK_kDD_IOCTL
};
```

In C++:

```
enum VDK::DeviceDispatchID
{
    VDK::kDD_Init,
    VDK::kDD_Activate,
    VDK::kDD_Open,
    VDK::kDD_Close,
    VDK::kDDSyncRead,
    VDK::kDDSyncWrite,
    VDK::kDD_IOCTL
};
```

DeviceDispatchUnion

A variable of the DeviceDispatchUnion type is passed as a parameter to the Name_DeviceDriver() function.

In C:

```
union VDK_DeviceDispatchUnion
{
    struct
    {
        void            **dataH;
        char            *flags; /* used for kDDOpen only */
    } OpenClose_t;

    struct
    {
        void            **dataH;
        VDK_Ticks       timeout;
        unsigned int    dataSize;
        char            *data;
    } ReadWrite_t;

    struct
    {
        void            **dataH;
        void            *command;
        char            *parameters;
    } IOCtl_t;
};
```

In C++:

```
union VDK::DeviceDispatchUnion
{
    struct
    {
        void          **dataH;
        char          *flags; /* used for kDDOpen only */
    } OpenClose_t;

    struct
    {
        void          **dataH;
        VDK::Ticks    timeout;
        unsigned int   dataSize;
        char          *data;
    } ReadWrite_t;

    struct
    {
        void          **dataH;
        void          *command;
        char          *parameters;
    } IOCtl_t;
};
```

DeviceFlagID

The DeviceFlagID type is used to store the unique identifier of a DeviceFlag.

In C/C++:

```
enum DeviceFlagID
{
    /* Defined by IDDE in the Vdk.h file. */
};
```

EventBitID

The EventBitID type is used to store the unique identifier of an EventBit. Total number of EventBits in a system is the size of a data word –1:

- fifteen bits for ADSP-219x DSPs
- thirty-one bits for ADSP-21xxx and ADSP-TSxxx DSPs

In C/C++:

```
enum EventBitID
{
    /* Defined by IDDE in the Vdk.h file. */
};
```

EventID

The EventID is used to store the unique identifier of an Event. Total number of Events in a system is the size of a data word –1:

- fifteen bits on ADSP-219x DSPs
- thirty-one bits on ADSP-21xxx and ADSP-TSxxx DSPs

In C/C++:

```
enum EventID
{
    /* Defined by IDDE in the Vdk.h file. */
};
```

EventData

The EventData type has two Bitfields (mask and values) and a Boolean ([All match | Any match] logic) data members.

In C:

```
typedef struct
{
    bool                allMatch;
    VDK_Bitfield        mask;
    VDK_Bitfield        values;
} VDK_EventData;
```

In C++:

```
typedef struct
{
    bool                allMatch;
    VDK::Bitfield       mask;
    VDK::Bitfield       values;
} VDK::EventData;
```

Priority

The Priority type is used to denote the scheduling priority level of a Thread:

- The highest priority is 1 (zero is reserved)
- The lowest priority is the size of a data word – 2.

For ADSP-219x —fourteen bits; for ADSP-21xxx and ADSP-TSxxx processor — thirty bits.

In C:

```
enum VDK_Priority
{
    VDK_kPriority1,
    VDK_kPriority2,
    VDK_kPriority3,
    ...
    VDK_kPriority14/30
};
```

In C++:

```
enum VDK::Priority
{
    VDK::kPriority1,
    VDK::kPriority2,
    VDK::kPriority3,
    ...
    VDK::kPriority14/30
};
```

SemaphoreID

The SemaphoreID type is used to store the unique identifier of a Semaphore.

In C/C++:

```
enum SemaphoreID
{
    /* Defined by the IDDE in the Vdk.h file. */
};
```

ThreadID

The ThreadID type is used to store the unique identifier of a Thread. The ThreadID is obtained either dynamically as the return value from the [CreateThread\(\)](#) function, or statically from the IDDE.

In C:

```
typedef unsigned int VDK_ThreadID;
```

In C++:

```
typedef unsigned int VDK::ThreadID;
```

ThreadStatus

The ThreadStatus type is used to enumerate the state of a Thread.

In C:

```
enum VDK_ThreadStatus
{
    VDK_kReady,
    VDK_kSemaphoreBlocked,
    VDK_kEventBlocked,
    VDK_kSemaphoreBlockedWithTimeout,
    VDK_kEventBlockedWithTimeout,
    VDK_kDeviceFlagBlocked
    VDK_kDeviceFlagBlockedWithTimeout
    VDK_kSleeping,
    VDK_kUnknown
};
```

In C++:

```
enum VDK::ThreadStatus
{
    VDK::kReady,
    VDK::kSemaphoreBlocked,
    VDK::kEventBlocked,
    VDK::kSemaphoreBlockedWithTimeout,
    VDK::kEventBlockedWithTimeout,
    VDK::kDeviceFlagBlocked
    VDK::kDeviceFlagBlockedWithTimeout
    VDK::kSleeping,
    VDK::kUnknown
};
```

ThreadType

The ThreadType is used to store the unique identifier of a Thread class.

In C:

```
enum ThreadType
{
    // Defined by IDDE in the Vdk.h file.
};
```

In C++:

```
enum ThreadType
{
    // Defined by IDDE in the Vdk.h file.
};
```

Ticks

Time is measured in system ticks. A Tick is the amount of time between hardware interrupts generated by a hardware timer.

In C:

```
typedef unsigned int VDK_Ticks;
```

In C++:

```
typedef unsigned int VDK::Ticks;
```

SystemError

The `SystemError` type is used to enumerate system-defined errors thrown to the error handler.

In C:

```
enum VDK_SystemError
{
    VDK_kUnknownThreadType = INT_MIN,
    VDK_kUnknownThread,
    VDK_kThreadCreationFailure,
    VDK_kUnknownSemaphore,
    VDK_kUnknownEventBit,
    VDK_kUnknownEvent,
    VDK_kInvalidPriority,
    VDK_kInvalidDelay,
    VDK_kSemaphoreTimeout,
    VDK_kEventTimeout,
    VDK_kBlockInInvalidRegion,
    VDK_kDbgPossibleBlockInRegion,
    VDK_kInvalidPeriod,
    VDK_kAlreadyPeriodic,
    VDK_kNonperiodicSemaphore,
    VDK_kDbgPopUnderflow,
    VDK_kDD_BadDeviceID,
    VDK_kDD_BadDeviceDescriptor,
    VDK_kDD_OpenFailure,
    VDK_kDD_CloseFailure,
    VDK_kDD_ReadFailure,
    VDK_kDD_WriteFailure,
    VDK_kDD_IOCTLFailure,
    VDK_kDD_InvalidDeviceFlag,
    VDK_kDD_DeviceTimeout,
    VDK_kNoError = 0,
    VDK_kFirstUserError,
    VDK_kLastUserError = INT_MAX
};
```

In C++:

```
enum VDK::SystemError
{
    VDK::kUnknownThreadType = INT_MIN,
    VDK::kUnknownThread,
    VDK::kThreadCreationFailure,
    VDK::kUnknownSemaphore,
    VDK::kUnknownEventBit,
    VDK::kUnknownEvent,
    VDK::kInvalidPriority,
    VDK::kInvalidDelay,
    VDK::kSemaphoreTimeout,
    VDK::kEventTimeout,
    VDK::kBlockInInvalidRegion,
    VDK::kDbgPossibleBlockInRegion,
    VDK::kInvalidPeriod,
    VDK::kAlreadyPeriodic,
    VDK::kNonperiodicSemaphore,
    VDK::kDbgPopUnderflow,
    VDK::kDD_BadDeviceID,
    VDK::kDD_BadDeviceDescriptor,
    VDK::kDD_OpenFailure,
    VDK::kDD_CloseFailure,
    VDK::kDD_ReadFailure,
    VDK::kDD_WriteFailure,
    VDK::kDD_IOCTLFailure,
    VDK::kDD_InvalidDeviceFlag,
    VDK::kDD_DeviceTimeout,
    VDK::kNoError = 0,
    VDK::kFirstUserError,
    VDK::kLastUserError = INT_MAX
};
```

VersionStruct

The constant VersionStruct is used to store four integers describing the system parameters:

- VDK API version number
- DSP family supported
- base DSP product supported
- build number

In C:

```
typedef struct
{
    int                mApiVersion;
    VDK_DSPFamilies    mFamily;
    VDK_DSPProduct     mDevice;
    long               mBuildNumber;
} VDK_VersionStruct;
```

In C++:

```
typedef struct
{
    int                mApiVersion;
    VDK::DSPFamilies   mFamily;
    VDK::DSPProduct    mDevice;
    long               mBuildNumber;
} VDK::VersionStruct;
```