SERVO**STAR**® MC/SC

API Reference Manual

SERVO**STAR**® MC/SC Product Series
API Reference Manual
M-MC-005-N
(Softcopy: M-MC-005-N)

Record of Manual Revisions

ISSUE NO.	DATE	BRIEF DESCRIPTION OF REVISION	API VERSION
0	3/1/1999	Preliminary issue for review	2.0.0
1	5/17/1999	Initial release	2.0.4
2	6/5/2000	Updated for TCP/IP support	2.1.0
3	10/22/2000	Minor corrections	3.0.4

Copyright Information

Contents

Chapter 1 Overview

Introduction

The Kollmorgen API addresses several concerns when interfacing your program with Kollmorgen drives and controllers including:

- sending commands (and receiving responses)
- reading and setting amplifier or controller variables
- sending and retrieving files
- error handling

The API addresses these concerns by providing a library of functions which allow the user to describe their system in terms of *devices* (amplifiers, controllers, axes, groups) and then communicate with these devices individually. Central to the API is the concept of a *device table.* The device table is a data store (database) which is managed by the API for the user. The device table is populated with devices by the programmer, allowing the programmer to execute commands on any device in the device table. Most details of the communication techniques used are hidden allowing the programmer to focus on high level programming instead of communication protocols.

General Information

The Kollmorgen API is available for Windows NT and is compatible with a wide variety of 32-bit programming tools including Microsoft Visual Basic and Microsoft Visual C++. The API is provided as a Windows DLL which makes it accessible from most Windows programming languages.

Most Kollmorgen API functions return an error code which indicates the success of the action requested by the programmer. Full text descriptions of the error are often associated with the error and can be accessed via API calls.

Initialization

The first step in using the API is to call the initialization function **KMInitialize**. This ensures that the information which the API maintains internally about your application is correct. No other API calls should be made before the API is initialized.

Device table creation

After initializing the API the next task is to describe your system to the API. For example if you have two ServoStar drives connected to your host system you need want to tell the API how they are connected (COM port, baud rate, etc) and assign a name to each of the drives. A more complicated system would contain a ServoStar MC motion controller with several ServoStar drives attached. Each of the drives as well as the controller must be described to the API.

This process involves describing and adding the devices you wish to control to the API library device table. The API contains a series of functions which allow the programmer to add devices to the API device table including **KMCreateSerialAxis**, **KMCreateSercosAxis**, **KMCreateController**. Each of the KMCreate functions returns a handle to the device which was just created. These handles are used to indicate to the rest of the API which device you are referring to. To remove devices from the device table use the **KMDestroyDevice** function.

Device tables can also be saved and loaded via the **KMWriteDeviceFile**, **KMReadDeviceFile** functions. One common source of device tables is ServoStar MotionLink which helps the user determine their system configuration.

Note for ServoStar MC users

The handle returned by **KMCreateSercosAxis** is a handle to the axis. An axis encompasses data on the drive and the controller. This will be important when you begin accessing data via the axis handle because you must select whether you are accessing the drive or the controller.

Accessing devices

The API provides two mechanisms for communicating with devices, 1) command execution and 2) variable access.

Command execution

Command execution is handled via two functions 1) **KMExecuteCmd**, 2) **KMExecuteCmdResponse**. Each of these functions take a handle to the device the command should be executed on and a string buffer which contains the command to be executed. **KMExecuteCmdResponse** also takes a user buffer for the response to the command and an indication of the length of the user buffer.

Variable access

Variable access has a few more functions than one would initially think. The list includes:

KMVariableDriveGetLongValue	KMVariableControllerGetLongValue
KMVariableDriveGetDblValue	KMVariableControllerGetDblValue
KMVariableDriveGetStringValue	KMVariableControllerGetStringValue
KMVariableDriveSetLongValue	KMVariableControllerSetLongValue
KMVariableDriveSetDblValue	KMVariableControllerSetDblValue
KMVariableDriveSetStringValue	KMVariableControllerSetStringValue

Notice there are function to get and set variables for each type (long, double, string) that is commonly used, as well as functions to access the two components of an axis (drive and controller). The KMVariableControllerXXX functions are only used when programming with a ServoStar MC.

Types

The differentiation between the different types is for user convenience. The user may access variables that are of type *long* on the MC with the **KMVariableControllerGetStringValue** function if desired. Obviously if the variable is of type *string* the **KMVariableControllerGetStringValue** function should be used.

Error messages

Most of the API functions return an error code. Successful API functions return **KM_ERR_OK**. Errors originate in the device itself or the API. When the message originates within the API an error number is returned which can be compared against the list of errors in the API header files (C/C++) or the global files (Visual Basic). In addition the API also assigns a text message to the error which can be retrieved by calling **KMErrorGetMessage**. When the error originates in the device (drive or controller) the API "parses" the message and stores the relevant information including the error message and the error number as given by the device. This information can be retrieved for the last error via **KMErrorGetDeviceMessage** similar to the Win32 API GetLastError function. Finally, the text of the error message, as the device originally sent, can be retrieved via **KMErrorGetOriginalDeviceMessage**.

Termination (cleanup)

The last call to the API before your application exits should be a call to **KMTerminate**. The functions ensures the API handles cleanup of the internal information it maintains about your application.

Groups

For controllers such as the ServoStar MC that support combining more than one axis into a *group* that API provides **KMCreateControllerGroup** which, not surprisingly, creates a handle referring to the group. The API also provides several functions which allow the programmer to indicate which axes below to the group (**KMAddAxisToGroup, KMRemoveAxisFromGroup**). Use **KMDestroyDevice** to remove a group from the device table.

Sending/Retrieving files

For the sending and retrieval of files the API provides users with two functions, **KMPutFile** and **KMGetFile**, respectively. In addition to the device handle parameter, the path of the file and the actual command must be sent when calling either of these functions.

Advanced device table access

The device table entry access functions can be used to retrieve a handle to a device or information about a particular device. For example, the API provides the function **KMGetAxisController** which returns a handle to the controller that an axis is associated with. Other functions perform actions such as retrieval of the name, ID, product type, communication type, and communication port (for serial devices) of a device.

In order to iterate through the device table, a device table iterator needs to be created using the **KMCreateDeviceIterator**. An iterator can be created to display all the controllers, axes, or groups or any combination of the three. Once created the functions **KMGetNextDevice** and **KMGetPrevDevice** may be called to move to the next or previous device, respectively, on the iterator's list.

Advanced asynchronous message handling

Some controllers and drives generate messages not related to a specific command called *asynchronous messages*. Some examples of asynchronous messages are overspeed warnings, limit switch closures and runtime error messages from ServoStar MC tasks. When these messages are received the API converts them into Windows messages (events) which the programmer can request to have delivered to their application.

By default the API displays each asynchronous message in a modal dialog box (via the Win32 MessageBox function). If the programmer wants to handle these messages differently the **KMAsyncSetHandler** API function allows a window to be *registered* as the destination for these messages. **KMAsyncGetHandler** can be used to save the previous error handler to that the programmer can restore it on demand.

Like all Windows messages, **WM_KM_ASYNC** has two parameters, wParam and lParam. lParam contains a handle to the buffer which contains the asynchronous message as received by the API. The programmer can use **KMAsyncGetMessage** to get the asynchronous message from the API.

Asynchronous messages and Microsoft Visual Basic

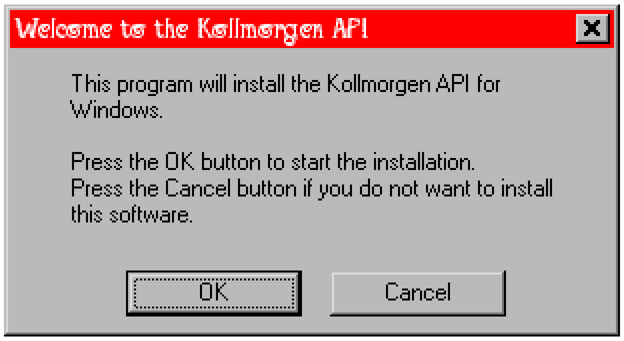
Visual Basic does not provide an easy mechanism for directly handling Windows messages. In order to overcome this limitation a custom control called KMAPI has been provided which has an event representing the receipt of an asynchronous message. Placing this control on your form automatically initializes the API (**KMInitialize**) and registers the form for asynchronous messages.

Chapter 2 Installing the API

Installation

In this chapter you will install the files required to use the Kollmorgen Application Program Interface (API). Before you use the API, you must install the API into the appropriate directory. Follow these steps to install the Kollmorgen API:

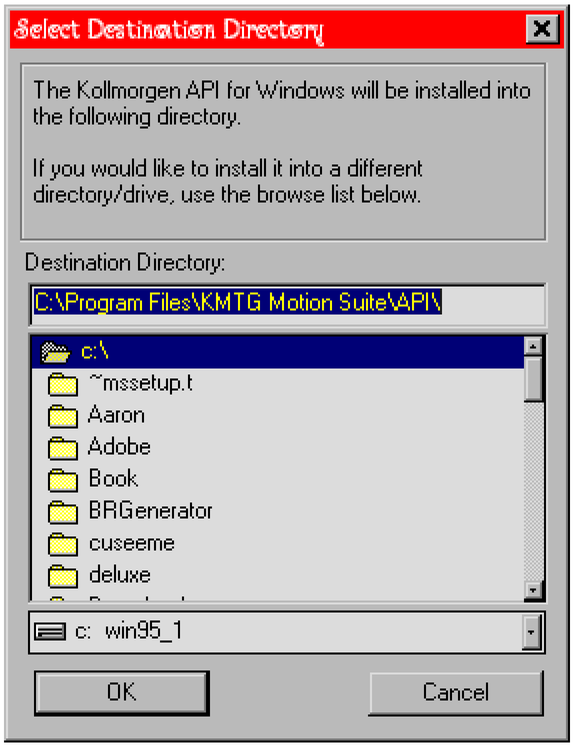
1. Insert floppy disk labeled "Kollmorgen API" into a: drive
2. Run a:\Setup.exe
3. Push Ok to continue with installation of the API when prompted with this screen.



This program will install the Kollmorgen API for Windows.

Press the OK button to start the installation.
Press the Cancel button if you do not want to install this software.

4. Choose the directory where you would like to install the API. You may create a new directory from this dialog as well as browse for an old one. After you have entered you directory select OK.



Select Destination Directory

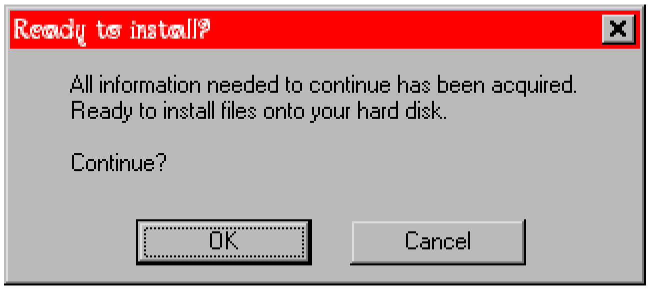
The Kollmorgen API for Windows will be installed into the following directory.

If you would like to install it into a different directory/drive, use the browse list below.

Destination Directory:

C:\Program Files\KMTG Motion Suite\API\

5. Select OK to continue installation when prompted with the following dialog.

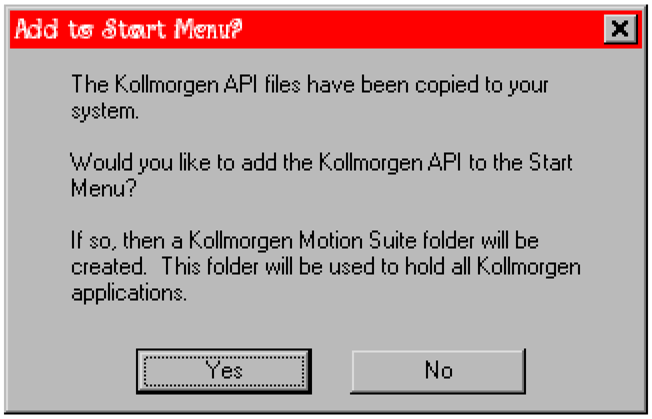


Ready to install?

All information needed to continue has been acquired.
Ready to install files onto your hard disk.

Continue?

6. Select yes here if you would like a link to the API added to your “Start” menu.

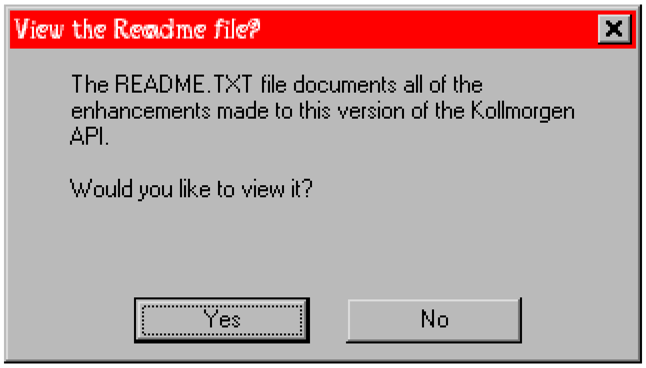

Add to Start Menu?

The Kollmorgen API files have been copied to your system.

Would you like to add the Kollmorgen API to the Start Menu?

If so, then a Kollmorgen Motion Suite folder will be created. This folder will be used to hold all Kollmorgen applications.

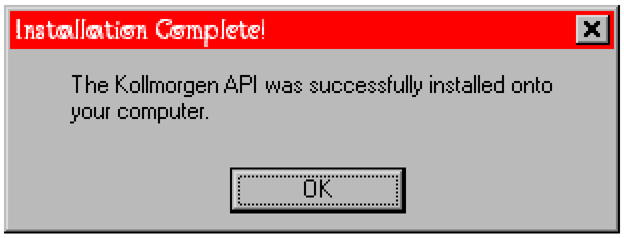
7. Select yes if you would like to read about the latest and greatest features the API has to offer.


View the Readme file?

The README.TXT file documents all of the enhancements made to this version of the Kollmorgen API.

Would you like to view it?

8. Select OK here to exit the installation program. The API has now been installed on your computer.


Installation Complete!

The Kollmorgen API was successfully installed onto your computer.

Chapter 3 Library Reference

API Initialization

KMInitialize

Prepares the API for use.

For C/C++:

KMErrorCode KMInitialize (void)

For Visual Basic:

KMInitialize() As Long

For C#:
System.Int32 IKMInitialize ()

Return Value	KM_ERR_OK if API is successfully initialized KM_ERR_WINDOWS_API if the asynchronous message window can't be created KM_ERR_MEM_LOCK if the application context was not successfully created
Remarks	The KMInitialize function must be called before any other API functions are called.
See Also	KMTerminate
Example	See Example #1

KMTerminate

Performs API cleanup required for proper application termination.

For C/C++:

KMErrorCode KMTerminate (void)

For Visual Basic:

KMTerminate() As Long

Fo**r C#:**
IKMTerminate ()

Return Value	KM_ERR_OK if API is successfully terminated KM_ERR_MEM_LOCK if the application context cannot be locked or removed KM_ERR_NO_CONTEXT if the application context cannot be found in the application context registry
Remarks	The **KMTerminate** function must be called before the application using the API terminates. **KMTerminate** can be called at any time in order to flush the present device table.
See Also	KMInitialize
Example	See Example #1

Device Management

KMCreateController

Creates a device table entry for a controller.

For C/C++:

KMDevice KMCreateController(KMProductType controllerProduct, LPSTR controllerName, KMCommType controllerCommType, short controllerID, LPSTR commOptions)

For Visual Basic:

KMCreateController (ByVal controllerProduct As Integer, ByVal controllerName$, ByVal controllerCommType As Integer, ByVal controllerID As Integer, ByVal commOptions$) As Long

For C#:

System.IntPtr IKMCreateController (System.UInt32 controllerProduct , System.String controllerName , System.UInt32 controllerCommType , System.UInt32 controllerID , System.String commOptions)

controllerProduct	Type of controller to create
controllerName	String name for the controller
controllerCommType	How the API should communicate with the controller (see communication type description)
controllerID	What the controller's address is
commOptions	COM port used by the controller

Return Value A handle referring to the controller.

Remarks See product type description for *controllerProduct***.**

controllerName is provided for user convenience and is not used internally by the API.

See communication type description for *controllerCommType.*

commOptions is a general parameter that is intended for many uses. Presently, only connections of type COMM_TCPIP use this parameter. If COMM_TCPIP is selected then the *commOptions* parameter allows the user to specify what controller to connect to by IP address, controller name, serial number or DIP switch setting. The format for *commOptions* is as follows:

- "IP: xx.xx.xx.xx" where xx.xx.xx.xx is an IP address (e.g. "192.10.34.6").
- "NM: yyyyy" where yyyyy is a name assigned by the user to the Sys.Name property of the controller.
- "SN: XXXXX-XXX" where XXXXX-XXX is the serial number assigned to the controller at the factory and printed on the model number label.
- "DP: XXXXXX" where XXXXXX is the value (in decimal) of the DIP switch setting on controllers that support. This is useful for user configuration of addresses.

When connecting via serial the IP address should is always be specified as 91.0.0.2.

If *commOptions* is not used, the parameter should be an empty string ("").

KMCreateController allocates memory on behalf of the user. This memory must be freed before program termination via **KMDestroyDevice**.

The range for *controllerDevice* is 1 - 9.

See Also KMCreateSercosAxis, KMCreateSerialAxis, KMDestroyDevice

Example See Example #1

KMCreateSercosAxis

Create a device table entry for a SERCOS axis and attach it to a controller.

For C/C++:

KMDevice KMCreateSercosAxis(KMProductType axisProduct, LPSTR axisName, KMDevice controllerDevice)

For Visual Basic:

KMCreateSercosAxis(ByVal axisProduct As Integer, ByVal axisName$, ByVal controllerDevice As Long) As Long

For C#:

System.IntPtr IKMCreateSercosAxis (System.UInt32 axisProduct , System.String axisName , System.IntPtr controllerDevice)

axisProduct	Type of axis to create
axisName	String name for the axis
controllerDevice	Handle of the controller the axis is attached to

Return Value A handle referring to the axis.

Remarks

KMCreateSercosAxis allocates memory on behalf of the user. This memory must be freed before program termination via **KMDestroyDevice**.

See product type description for *axisProduct*.

The API uses *axisName* to access variables on the axis, therefore it must match the name as it is set within the controller.

controllerDevice is the handle of the controller the axis is physically connected to.

See Also KMCreateController, KMCreateSerialAxis, KMDestroyDevice

Example See Example #1

KMCreateSerialAxis

Create a device table entry for a serial axis.

For C/C++:

KMDevice KMCreateSerialAxis (KMProductType axisProduct, LPSTR axisName, KMCommType commType, LPSTR commPort)

For Visual Basic:

KMCreateSerialAxis (ByVal axisProduct As Integer, ByVal axisName$, ByVal commType As Integer, ByVal commPort$) As Long

For C#:
System.IntPtr IKMCreateSerialAxis (System.UInt32 axisProduct , System.String axisName , System.UInt32 commType , System.String commPort)

axisProduct	Type of axis to create
axisName	String name for the axis
commType	How to communicate with the axis
commPort	COM port the drive is attached to

Return Value KM_ERR_OK if command is successfully transmitted

Remarks *axisName* must match the multidrop address of the amplifier.

See Also KMCreateController, KMCreateSerialAxis, KMDestroyDevice

Example See Example #1

KMDestroyDevice

Free memory that was allocated by the API for the device.

For C/C++:

KMErrorCode KMDestroyDevice(KMDevice device)

For Visual Basic:

KMDestroyDevice(ByVal device As Long) As Long

For C#:

System.Int32 IKMDestroyDevice (System.IntPtr device)

device	Handle for the device to be destroyed
Return Value	KM_ERR_OK if command is successfully transmitted
Remarks	The device handle will not be valid after calling **KMDestroyDevice**.
See Also	KMCreateController, KMCreateSercosAxis, KMCreateSerialAxis
Example	See Example #1

Product Types

The following constants are valid product types:

Constant	Value
PROD_NONE	0
PROD_SERVOSTAR	1
PROD_SERCOS_SERVOSTAR	2
PROD_SSMC	3
PROD_SSMC_DRIVER	4

Product Classes

The following constants are valid product classes:

Constant	Value
CLASS_NONE	0
CLASS_AXIS	1
CLASS_GROUP	2
CLASS_CONTROLLER	3

Communication Types

The following constants are valid communication types:

Constant	Value
COMM_NONE	0
COMM_SERIAL	1
COMM_SERIAL_MULTI	2
COMM_SERCOS	3
COMM_PLUGIN	4
COMM_TCPIP	5

Device Access

KMExecuteCmd

Sends a command to a device.

For C/C++:

KMErrorCode KMExecuteCmd(KMDevice device, LPSTR cmdStr)

For Visual Basic:

KMExecuteCmd(ByVal device As Long, ByVal cmdStr$) As Long

For C#:
System.Int32 IKMExecuteCmd (System.IntPtr device , System.String cmdStr)

device	Handle for the device the command is being sent to.
cmdStr	The command to send.

Return Value KM_ERR_OK if command is successfully transmitted

Remarks Sends the command *cmdStr* unmodified to the device specified by *device.* Should be used with commands that do not have responses.

See Also KMExecuteCmdResponse, KMErrorGetMessage, KMErrorGetDeviceMessage

Example See Example #1

KMExecuteCmdResponse

Sends a command to a device and waits for a response.

For C/C++:

KMErrorCode KMExecuteCmdResponse(KMDevice device, LPSTR cmdStr, LPSTR outStr, DWORD outSize)

For Visual Basic:

KMExecuteCmdResponse(ByVal device As Long, ByVal cmdStr$, ByVal outStr$, ByVal outSize As Long) As Long

For C#:

System.Int32 IKMExecuteCmdResponse (System.IntPtr device , System.String cmdStr , char[] outStr , System.UInt32 outSize)

device	Handle for the device the command is being sent to.
cmdStr	The command to send.
outStr	A buffer for the response.
outSize	Size of the *outStr* buffer.

Return Value KM_ERR_OK if command is successfully transmitted

Remarks Sends the command *cmdStr* unmodified to the device specified by *device.* Should be used with commands that have responses.

See Also KMExecuteCmd, KMErrorGetMessage, KMErrorGetDeviceMessage

Example See Example #1

KMVariableControllerGetLongValue

Gets the contents of a variable of type long from a controller.

For C/C++:

KMErrorCode KMVariableControllerGetLongValue (KMDevice device, LPSTR varName, LPLONG pValue)

For Visual Basic:

KMVariableControllerGetLongValue(ByVal device As Long, ByVal varName$, pValue As Long) As Long

For C#:

System.Int32 IKMVariableControllerGetLongValue (System.IntPtr device, System.String varName , System.UInt32 pValue)

device	Handle for the device that contains the variable.
varName	Name of the variable.
pValue	Pointer to a long integer to receive the value of the device variable.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue

Example See Example #1

KMVariableDriveGetLongValue

Gets the contents of a variable of type long from an axis.

For C/C++:

KMErrorCode KMVariableDriveGetLongValue (KMDevice device, LPSTR varName, LPLONG pValue)

For Visual Basic:

KMVariableDriveGetLongValue(ByVal device As Long, ByVal varName$, pValue As Long) As Long

For C#:

System.Int32 IKMVariableDriveGetLongValue (System.IntPtr device , System.String varName , System.UInt32 pValue)

device	Handle for the device that contains the variable.
varName	Name of the variable.
pValue	Pointer to a long integer to receive the value of the device variable.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue

Example See Example #1

KMVariableControllerGetDblValue

Gets the contents of a variable of type double from a controller.

For C/C++:

KMErrorCode KMVariableControllerGetDoubleValue (KMDevice device, LPSTR varName, LPDOUBLE pValue)

For Visual Basic:

KMVariableControllerGetDblValue(ByVal device As Long, ByVal varName$, pValue As Double) As Long

For C#:
System.Int32 IKMVariableControllerSetDblValue (System.IntPtr device , System.String varName , System.Double pValue)

device	Handle for the device that contains the variable.
varName	Name of the variable.
pValue	Pointer to a double to receive the value of the device variable.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetLongValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue

Example See Example #1

KMVariableDriveGetDblValue

Gets the contents of a variable of type double from an axis.

For C/C++:

KMErrorCode KMVariableDriveGetDoubleValue (KMDevice device, LPSTR varName, LPDOUBLE pValue)

For Visual Basic:

KMVariableDriveGetDblValue(ByVal device As Long, ByVal varName$, pValue As Double) As Long

For C#:
System.Int32 IKMVariableDriveGetDblValue (System.IntPtr device , System.String varName , System.Double pValue)

device	Handle for the device that contains the variable.
varName	Name of the variable.
pValue	Pointer to a double to receive the value of the device variable.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue

Example See Example #1

KMVariableControllerGetStringValue

Gets the contents of a variable of type string from a controller.

For C/C++:

KMErrorCode KMVariableControllerGetStringValue (KMDevice device, LPSTR varName, LPSTR pValue, long valueLen)

For Visual Basic:

KMVariableControllerGetStringValue(ByVal device As Long, ByVal varName$, ByVal pValue$, ByVal valueLen As Long) As Long

For C#:

System.Int32 IKMVariableControllerGetStringValue (System.IntPtr device, System.String varName , char[] pValue , System.UInt32 valueLen)

device	Handle for the device that contains the variable.
varName	Name of the variable.
pValue	Pointer to a character buffer to receive the value of the device variable.
valueLen	Length of the character buffer *pValue*.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue

Example See Example #1

KMVariableDriveGetStringValue

Gets the contents of a variable of type string from an axis.

For C/C++:

KMErrorCode KMVariableDriveGetStringValue (KMDevice device, LPSTR varName, LPSTR pValue, long valueLen)

For Visual Basic:

KMVariableDriveGetStringValue(ByVal device As Long, ByVal varName$, ByVal pValue$, ByVal valueLen As Long) As Long

For C#:

System.Int32 IKMVariableDriveGetStringValue (System.IntPtr device, System.String varName , char[] pValue , System.UInt32 valueLen)

device	Handle for the device that contains the variable.
varName	Name of the variable.
pValue	Pointer to a character buffer to receive the value of the device variable.
valueLen	Length of the character buffer *pValue*.

Return Value	KM_ERR_OK on success
Remarks	varName is the name of the variable as it is on the device.
See Also	KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue
Example	See Example #1

KMVariableControllerSetLongValue

Sets the contents of a variable in a controller of type long.

For C/C++:

**KMErrorCode KMVariableControllerSetLongValue (KMDevice device,
LPSTR varName, long value)**

For Visual Basic:

**KMVariableControllerSetLongValue(ByVal device As Long, ByVal varName$,
ByVal value As Long) As Long**

For C#:
**System.Int32 IKMVariableControllerSetLongValue (System.IntPtr device ,
System.String varName , System.UInt32 Value)**

device	Handle for the device that contains the variable.
varName	Name of the variable.
value	Value to set the variable *varName* to.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetLongValue,
KMVariableControllerGetDblValue,
KMVariableControllerGetStringValue,
KMVariableControllerSetDblValue,
KMVariableControllerSetStringValue,
KMVariableDriveGetLongValue,
KMVariableDriveGetDblValue,
KMVariableDriveGetStringValue,
KMVariableDriveSetLongValue,
KMVariableDriveSetDblValue,
KMVariableDriveSetStringValue

Example See Example #1

KMVariableDriveSetLongValue

Sets the contents of a variable in an axis of type long.

For C/C++:

KMErrorCode KMVariableDriveSetLongValue (KMDevice device, LPSTR varName, long value)

For Visual Basic:

KMVariableDriveSetLongValue(ByVal device As Long, ByVal varName$, ByVal value As Long) As Long

For C#:
System.Int32 IKMVariableDriveSetLongValue (System.IntPtr device , System.String varName , System.UInt32 value)

device	Handle for the device that contains the variable.
varName	Name of the variable.
value	Value to set the variable *varName* to.

Return Value	KM_ERR_OK on success
Remarks	varName is the name of the variable as it is on the device.
See Also	KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue
Example	See Example #1

KMVariableControllerSetDblValue

Sets the contents of a variable in a controller of type double.

For C/C++:

KMErrorCode KMVariableControllerSetDblValue (KMDevice device, LPSTR varName, double value)

For Visual Basic:

KMVariableControllerSetDblValue(ByVal device As Long, ByVal varName$, ByVal value As Double) As Long

For C#:

System.Int32 IKMVariableControllerSetDblValue (System.IntPtr device , System.String varName , System.Double value)

device	Handle for the device that contains the variable.
varName	Name of the variable.
value	Value to set the variable *varName* to.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue

Example See Example #1

KMVariableDriveSetDblValue

Sets the contents of a variable in an axis of type double.

For C/C++:

KMErrorCode KMVariableDriveSetLongValue (KMDevice device, LPSTR varName, double value)

For Visual Basic:

KMVariableDriveSetDblValue(ByVal device As Long, ByVal varName$, ByVal value As Double) As Long

For C#:
System.Int32 IKMVariableDriveSetDblValue (System.IntPtr device , System.String varName , System.Double value)

device	Handle for the device that contains the variable.
varName	Name of the variable.
value	Value to set the variable *varName* to.

Return Value	KM_ERR_OK on success
Remarks	varName is the name of the variable as it is on the device.
See Also	KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetStringValue
Example	See Example #1

KMVariableControllerSetStringValue

Sets the contents of a variable in a controller of type string.

For C/C++:

KMErrorCode KMVariableControllerSetStringValue (KMDevice device, LPSTR varName, LPSTR pValue)

For Visual Basic:

KMVariableControllerSetStringValue(ByVal device As Long, ByVal varName$, ByVal value$) As Long

For C#:
System.Int32 IKMVariableControllerSetStringValue (System.IntPtr device , System.String varName , System.String value)

device	Handle for the device that contains the variable.
varName	Name of the variable.
pValue	Pointer to a character buffer to set the variable *varName* to.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue, KMVariableDriveSetStringValue

Example See Example #1

KMVariableDriveSetStringValue

Sets the contents of a variable in an axis of type string.

For C/C++:

KMErrorCode KMVariableDriveSetStringValue (KMDevice device, LPSTR varName, LPSTR pValue)

For Visual Basic:

KMVariableDriveSetStringValue(ByVal device As Long, ByVal varName$, ByVal value$) As Long

For C#:
System.Int32 IKMVariableDriveSetStringValue (System.IntPtr device , System.String varName , System.String value)

device	Handle for the device that contains the variable.
varName	Name of the variable.
pValue	Pointer to a character buffer to set the variable *varName* to.

Return Value KM_ERR_OK on success

Remarks varName is the name of the variable as it is on the device.

See Also KMVariableControllerGetLongValue, KMVariableControllerGetDblValue, KMVariableControllerGetStringValue, KMVariableControllerSetLongValue, KMVariableControllerSetDblValue, KMVariableControllerSetStringValue, KMVariableDriveGetLongValue, KMVariableDriveGetDblValue, KMVariableDriveGetStringValue, KMVariableDriveSetLongValue, KMVariableDriveSetDblValue

Example See Example #1

Example #1

This program shows how to initialize the API, create devices and access variables on each of the devices.

C / C++

```
/* Declare device handles */
KMDevice devController;
KMDevice axisA1;

/*Declare variables */
long valueL;
double valueD;
char valueS[100];

/* Initialize the API */
KMInitialize();

/* Create a controller */
devController = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "");

/* Create axes */
axisA1 = KMCreateSercosAxis( PROD_SERVOSTAR, "A1", devController);

/* Move A1 */
KMExecuteCmd( devController, “MOVE A1 100”);

/* Get variables */
KMVariableControllerGetLongValue(axisA1,"VCRUISE",&valueL);
KMVariableControllerGetDblValue(axisA1,"PFINAL",&valueD);
KMVariableControllerGetStringValue(axisA1,”AMAX”,valueS,sizeof(valueS)
);

/* Set variables */
KMVariableControllerSetLongValue(axisA1,"VCRUISE",valueL);
KMVariableControllerSetDblValue(axisA1,"PFINAL",valueD);
KMVariableControllerSetStringValue(axisA1,”AMAX”,valueS);

/* Destroy device table entries */
KMDestroyDevice(axisA1);
KMDestroyDevice(devController);

/* Terminate the API */
KMTerminate();
```

Visual Basic

```
'Declare device handles
Dim devController As Long
Dim axisA1 As Long

'Declare variables
Dim valueL As Long
Dim valueD As Double
Dim errorL As Long

'Initialize the API
errorL = KMInitialize()

'Create a controller
devController = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "")

'Create axes
axisA1 = KMCreateSercosAxis( PROD_SERVOSTAR, "A1", devController)

'Move A1
errorL = KMExecuteCmd( devController, "MOVE A1 100");

'Get variables
errorL = KMVariableControllerGetLongValue(axisA1,"VCRUISE",valueL)
errorL = KMVariableControllerGetDblValue(axisA1,"PFINAL",valueD)
errorL = KMVariableControllerGetStringValue(axisA1,"AMAX",valueS,
Length(valueS))

'Set variables
errorL = KMVariableControllerSetLongValue(axisA1,"VCRUISE",valueL)
errorL = KMVariableControllerSetDblValue(axisA1,"PFINAL",valueD)
errorL = KMVarliableControllerSetStringValue(axisA1,"AMAX",valueS)

'Destroy device table entries
errorL = KMDestroyDevice(axisA1)
errorL = KMDestroyDevice(devController)

'Terminate the API
errorL = KMTerminate()
```

KMGetFile

Retrieve a file from a device.

For C/C++:

KMErrorCode KMGetFile(KMDevice device, LPSTR cmdStr, LPSTR filename)

For Visual Basic:

KMGetFile(ByVal device As Long, ByVal cmdStr$, ByVal filename$) As Long

For C#:
System.Int32 IKMGetFile (System.IntPtr device , System.String cmdStr , System.String fileName)

device	Handle for the device the file is being retrieved from.
cmdStr	The command string to cause the device to send a file.
filename	Filename to save file to on host computer.

Return Value	KM_ERR_OK if command is successfully transmitted
Remarks	The filename (if there is one) in *cmdStr* and *filename* parameters do not have to match.
See Also	KMPutFile
Example	See Example #2

KMPutFile

Sends a file to a device.

For C/C++:

KMErrorCode KMPutFile(KMDevice device, LPSTR cmdStr, LPSTR filename)

For Visual Basic:

KMPutFile(ByVal device As Long, ByVal cmdStr$, ByVal filename$) As Long

For C#:
System.Int32 IKMPutFile (System.IntPtr device , System.String cmdStr , System.String fileName)

device	Handle for the device the file is being sent to.
cmdStr	The command string to cause the device to retrieve a file.
filename	Filename of file on host computer.

Return Value KM_ERR_OK if command is successfully transmitted

Remarks The filename (if there is one) in *cmdStr* and *filename* parameters do not have to match.

See Also KMGetFile

Example See Example #2

Example #2

This program shows how to send and retrieve files to/from a device (controller or amplifier) via the API.

C / C++

```
/* Declare device handles */
KMDevice devController;

/* Initialize the API */
KMInitialize();

/* Create a controller */
devController = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "");

/* Get a file called “PROG1.PRG” from devController */
KMGetFile(devController,”RETRIEVE PROG1.PRG”,”C:\\PROG1.PRG”);

/* Send a file called “PROG2.PRG” to devController */
KMPutFile(devController,”SEND PROG2.PRG”,”C:\\PROG2.PRG”);

/* Destroy device table entries */
KMDestroyDevice(devController);

/* Terminate the API */
KMTerminate();
```

Visual Basic

```
'Declare device handles
Dim devController As Long

'Declare Variables
Dim errorL as KMErrorCode

'Initialize the API
errorL = KMInitialize()

'Create a controller
devController = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "")

'Get a file called “PROG1.PRG” from devController
errorL = KMGetFile(devController,”RETRIEVE PROG1.PRG”,”C:\\PROG1.PRG”)

'Send a file called “PROG2.PRG” to devController
errorL = KMPutFile(devController,”SEND PROG2.PRG”,”C:\\PROG2.PRG”)

'Destroy device table entries
errorL = KMDestroyDevice(devController)

'Terminate the API
Dim errorL = KMTerminate()
```

KMDeviceSetCommProperties

Used to set the communications properties for a device.

For C/C++:

KMErrorCode KMDeviceSetCommProperties(KMDevice device, short baudRate, short byteSize, short parity, short stopBits);

For Visual Basic:

KMDeviceSetCommProperties(ByVal device As Long, ByVal baudRate As Integer, ByVal byteSize As Integer, ByVal parity As Integer, ByVal stopBits As Integer) As Long

For C#:

System.Int32 IKMDeviceSetCommProperties (System.IntPtr device , System.UInt32 baudRate , System.UInt32 byteSize , System.UInt32 parity , System.UInt32 stopBits)

device	handle of the device to be modified
baudRate	baud rate of the port
byteSize	byte size to be used with the port
parity	parity setting
stopBits	stopBits setting

Return Value	KM_ERR_OK if command is successfully transmitted
Remarks	This function must be passed a valid handle to a device.

See Also

Example

KMDeviceSetCheckSum

Turns the checksum on or off for a particular device.

For C/C++:

KMErrorCode KMDeviceSetCheckSum(KMDevice device, BOOL checkSumOn);

For Visual Basic:

KMDeviceSetCheckSum(ByVal device As Long, ByVal checkSumOn As Integer) As Long

For C#:
System.Int32 IKMDeviceSetCheckSum (System.IntPtr device , System.UInt32 checkSumOn)

device	handle of the device to be adjusted
checkSumOn	true for checksum on, false for checksum off

Return Value KM_ERR_OK if command is successfully transmitted

Remarks This function must be passed a valid handle to a device.

See Also

Example

KMDeviceSetEcho

Turns the echo on or off for a particular device.

For C/C++:

KMErrorCode KMDeviceSetEcho(KMDevice device, BOOL echoOn);

For Visual Basic:

KMDeviceSetEcho(ByVal device As Long, ByVal echoOn As Integer) As Long

For C#:

```
System.Int32 IKMDeviceSetEcho ( System.IntPtr device ,
                                System.UInt32 echoOn )
```

device	handle of the device to be adjusted
echoOn	true for echo on, false for echo off

Return Value KM_ERR_OK if command is successfully transmitted

Remarks This function must be passed a valid handle to a device.

See Also

Example

KMDeviceSetPrompt

Turns the prompt on or off for a particular device.

For C/C++:

KMErrorCode KMDeviceSetPrompt(KMDevice device, BOOL promptOn);

For Visual Basic:

KMDeviceSetPrompt(ByVal device As Long, ByVal promptOn As Integer) As Long

For C#:
System.Int32 IKMDeviceSetPrompt (System.IntPtr device , System.UInt32 promptOn)

device	handle of the device to be adjusted
promptOn	true for prompt on; false for prompt off

Return Value KM_ERR_OK if command is successfully transmitted

Remarks This function must be passed a valid handle to a device.

See Also

Example

KMGetMaxControllerAxis

Gets the maximum number of axis allowed for a given controller.

For C/C++:

short KMGetMaxControllerAxis(KMDevice controller);

For Visual Basic:

KMGetMaxControllerAxis(ByVal controller As Long) As Integer

For C#:
IKMGetMaxControllerAxis (System.IntPtr controller)

controller	handle of the controller to be checked
Return Value	maximum number of axis allowed
Remarks	This function must be passed a valid handle to a controller.
See Also	
Example	

Error Handling

Error handling is one of the most important elements to creating a robust application. The Kollmorgen API provides comprehensive error handling which allows the programmer to properly handle all situations. While using the API errors can occur at the following levels:

- Internal to the API

- host computer runs out of memory

- Communication between the API and the device

- device fails to respond in time

- Internal to the device

- a move cannot be made due to a limit switch being open

Every function in the API returns a result, which is almost always of the type KMErrorCode (Long for Visual Basic users). Each error type is well identified in the error code. The following is an example of the proper method of handling errors returned from the API:

C / C++

```
if ( err != KM_ERR_OK )
{
   char strBuf[1024];
   KMErrorCode err2;

   if ( IS_DEVICE_ERROR(err) ) // the error is device related
   {
      // This next call will eventually be KMErrorGetDeviceMessage,
      // but this function is not implemented yet.
      err2 = KMErrorGetOriginalDeviceMessage(strBuf, sizeof(strBuf));
      // insert your code to print the error message here
      if (err2 == KM_ERR_OK)
      {
         // insert your code to print the error message here
      }
      else
         printf("Error calling KMErrorGetOriginalDeviceMessage"
            "[%08X].", err2);
   }
   else
   {
      // the error is an API error so get the error message
      // from the API
      err2 = KMErrorGetMessage(err, strBuf, sizeof(strBuf));
      if (err2 == KM_ERR_OK)
      {
         // insert your code to print the error message here
      }
      else
         printf("Error calling KMErrorGetMessage [%08X].", err2);
   }
}
```

Visual Basic

```
Dim strBuf as String * 1024
Dim err as Long
Dim err2 as Long

If err <> KM_ERR_OK Then
    If IS_DEVICE_ERROR(err) Then  ' the error is device related
        ' This next call will eventually be KMErrorGetDeviceMessage,
        ' but this function is not implemented yet.
        err2 = KMErrorGetOriginalDeviceMessage(strBuf, 1024)
        ' insert your code to print the error message here
        If err2 = KM_ERR_OK Then
            ' insert your code to print the error message here
        Else
            MsgBox "Error calling KMErrorGetOriginalDeviceMessage"
        End If
    Else
        ' the error is an API error so get the error message
        ' from the API
        err2 = KMErrorGetMessage(err, strBuf, sizeof(strBuf))
        If err2 = KM_ERR_OK
            ' insert your code to print the error message here
        Else
            MsgBox "Error calling KMErrorGetMessage"
        End If
    End If
End If
```

Error Codes

The following is a list of the error codes that can be returned by the API.

Error Code	Description	Value (hex)
KM_ERR_OK		H00000000
KM_ERR_BAD		HFFFFFFFF
KM_ERR_BAD_CRC		H00800001
KM_ERR_TIME_OUT		H00800002
KM_ERR_BAD_DEVICE_ID		H00800003
KM_ERR_DEVICE_NOT_INSTALLED		H00800004
KM_ERR_LOCK_FAILED		H00800005
KM_ERR_NO_DATA		H00800006
KM_ERR_BUFFER_TOO_SMALL		H00800007
KM_ERR_ASYNC		H00800008
KM_ERR_FAIL_SEND_MSG		H00800009
KM_ERR_FAIL_ACK_NAK		H0080000A
KM_ERR_FAIL_ACCEPT		H0080000B
KM_ERR_FAIL_PROTOCOL		H0080000C
KM_ERR_FAIL_RESPOND		H0080000D
KM_ERR_UNKNOWN_MESSAGE		H0080000E
KM_ERR_MEM_LOCK		H0080000F
KM_ERR_FAIL_CREATE_FILE		H00800010
KM_ERR_FAIL_OPEN_FILE		H00800011
KM_ERR_FAIL_FIND_FILE		H00800012
KM_ERR_UNEXPECTED_EOF		H00800013
KM_ERR_NO_CONTEXT		H00800014
KM_ERR_MEM_ALLOC		H00800015
KM_ERR_INVALID_PRODUCT		H00800016
KM_ERR_SYSTEM		H00800017
KM_ERR_FAIL_FIND_NAME		H00800018
KM_ERR_INVALID_TYPE		H00800019
KM_ERR_NO_DEFAULT		H0080001A
KM_ERR_NO_MAX_MIN		H0080001B
KM_ERR_BAD_IRQ_NUMBER		H0080001C
KM_ERR_INVALID_FORMAT		H0080001D
KM_ERR_NOT_IMPLEMENTED		H0080001E
KM_ERR_FAIL_FIND_MSG		H0080001F
KM_ERR_NOT_DEVICE		H00800020
KM_ERR_FAILED_OPEN_DEVICE		H00800021
KM_ERR_INVALID_VALUE		H00800022
KM_ERR_INVALID_DEVICE_CONTEXT		H00800023
KM_ERR_GROUP_MAX_EXCEEDED		H00800024

KM_ERR_AXIS_ALREADY_IN_GROUP	H00800025
KM_ERR_FAIL_FIND_DEVICE	H00800026
KM_ERR_FAIL_WRITE_FILE	H00800027
KM_ERR_FAIL_READ_FILE	H00800028
KM_ERR_CONFIG_ALREADY_EXISTS	H00800029
KM_ERR_DEVICE_ALREADY_EXISTS	H0080002A
KM_ERR_FAIL_PRODUCT_DLL	H0080002B
KM_ERR_WINDOWS_API	H0080002C
KM_ERR_VARIABLE_NOT_FOUND	H0080002D
KM_ERR_SERIAL_FRAMING	H0080002E
KM_ERR_FAILED_CLOSE_DEVICE	H0080002F
KM_ERR_FAIL_PRODUCT_DLL	H00800030
KM_ERR_SYNTAX_ERROR	H00800100
KM_ERR_SSMC_ERROR	H00030027
KM_ERR_SERVOSTAR_ERROR	H00010027
KM_ERR_SSMC_DRIVER_ERROR	H00040027

KMErrorGetMessage

Get text of the error message related to an error code.

For C/C++:

KMErrorCode KMErrorGetMessage(KMErrorCode errCode, LPSTR buf, short bufLen)

For Visual Basic:

KMErrorGetMessage(ByVal errCode As Long, ByVal buf$, ByVal bufLen As Integer) As Long

For C#:
System.Int32 IKMErrorGetMessage (System.Int32 errCode , char[] buf , System.UInt32 bufLen)

errCode	Error code to get the message for
buf	Character array buffer to store error message in
bufLen	Length of *buf* array.

Return Value	KM_ERR_OK if command is successfully transmitted
Remarks	The text returned is device dependent and is not translated in any way. If more than one line of text is returned the lines will be separated with newline characters ('\n').
See Also	KMErrorGetDeviceMessage, KMErrorGetOriginalDeviceMessage
Example	See Example #3

KMErrorGetDeviceMessage

Get original text and error number of last device error.

For C/C++:

KMErrorCode KMErrorGetDeviceMessage(KMErrorCodePtr pErrCode, LPSTR buf, short bufLen)

For Visual Basic:

KMErrorGetDeviceMessage(pErrCode As Long, ByVal buf$, ByVal bufLen As Integer) As Long

For C#:

System.Int32 IKMErrorGetDeviceMessage (System.IntPtr pErrCode , char[] buf , System.UInt32 bufLen)

pErrCode	Pointer to variable to store last error number in
buf	Character array buffer for device error message
bufLen	Length of *buf* array

Return Value KM_ERR_OK if command is successfully transmitted

Remarks Last device error is stored on a per application context basis.

The text returned is device dependent and is not translated in any way. If more than one line of text is returned the lines will be separated with newline characters ('\n').

See Also KMErrorGetMessage, KMErrorGetOriginalDeviceMessage

Example See Example #3

KMErrorGetOriginalDeviceMessage

Retrieves entire original error message from buffer.

For C/C++:

KMErrorCode KMErrorGetOriginalDeviceMessage(LPSTR buf, short bufLen);

For Visual Basic:

KMErrorGetOriginalDeviceMessage(ByVal buf$, ByVal bufLen As Integer) As Long

For C#:
System.Int32 IKMErrorGetOriginalDeviceMessage (char[] buf , System.UInt32 bufLen)

buf	Character array buffer to store error message in
bufLen	Length of *buf* array.

Return Value	KM_ERR_OK if command is successfully transmitted
Remarks	The text returned is the original error message that was generated and stored in the buffer.
See Also	KMErrorGetDeviceMessage, KMErrorGetMessage
Example	See Example #3

Example #3

This program shows how to get the text descriptions associated with any error number or the last error that occurred.

C / C++

```
/* Declare device handles */
KMDevice devController;

/* Declare error code variables */
KMErrorCode err;
char strErrBuf[500];

/* Initialize the API */
KMInitialize();

/* Create a controller */
devController = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "");

/* A statement that will cause an error */
err = KMVariableGetLongValue(axisA1,"VCRUIS",&valueL); /* VCRUIS
should be VCRUISE */
if (err != KM_ERR_OK) /* do something if there was an error */
{
   KMErrorGetMessage(err,strErrBuf,sizeof(strErrBuf));
   printf(“The error message was: %s\n”,strErrBuf);
}

/* Another way to get the last error */
err = KMVariableGetLongValue(axisA1,"VCRUIS",&valueL); /* VCRUIS
should be VCRUISE */
KMErrorGetDeviceMessage(&err,strErrBuf,sizeof(strErrBuf));
if (err != KM_ERR_OK) /* do something if there was an error */
{
   printf(“The error message was: %s\n”,strErrBuf);
}

/* Destroy device table entries */
KMDestroyDevice(devController);

/* Terminate the API */
KMTerminate();
```

Visual Basic

```
'Declare device handles
Dim devController As Long

'Declare error code variables
Dim err As Long
Dim strErrBuf as String[10000]

'Initialize the API
err = KMInitialize()

'Create a controller
devController = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "")

'A statement that will cause an error
err = KMVariableGetLongValue(axisA1,"VCRUIS",valueL)
'VCRUIS should be VCRUISE
If err != KM_ERR_OK Then 'do something if there was an error

   err = KMErrorGetMessage(err,strErrBuf,10000)
   Print "The error message was: "; strErrBuf

EndIf

'Another way to get the last error
err = KMVariableGetLongValue(axisA1,"VCRUIS",valueL)
'VCRUIS should be VCRUISE
err = KMErrorGetDeviceMessage(err,strErrBuf,10000)
If err != KM_ERR_OK Then 'do something if there was an error

   Print "The error message was: "; strErrBuf

EndIf

'Destroy device table entries
err = KMDestroyDevice(devController)

'Terminate the API
err = KMTerminate()
```

Device Table Access

KMGetAxisController

Get a handle to the controller the axis is associated with.

For C/C++:

KMDevice KMGetAxisController(KMDevice axis)

For Visual Basic:

KMGetAxisController(ByVal axis As Long) As Long

For C#:
System.Int32 IKMGetAxisController (System.IntPtr axis)

axis	Handle to a axis
Return Value	Handle of the controller the axis is located on.
Remarks	An axis is assigned to a controller during creation.
See Also	KMCreateSercosAxis, KMCreateSerialAxis, KMCreateController
Example	See Example #4

KMGetDeviceName

Get the name assigned to the device.

For C/C++:

KMErrorCode KMGetDeviceName(KMDevice device, LPSTR name, long nameLen)

For Visual Basic:

KMGetDeviceName(ByVal device As Long, ByVal deviceName$, ByVal nameLen As Long) As Long

For C#:

System.Int32 IKMGetDeviceName (System.IntPtr device , System.String name , System.UInt32 nameLen)

device	Handle to a device
name	String buffer to place the name in
nameLen	Length of *name* buffer including zero terminator

Return Value KM_ERR_OK if command is successfully transmitted

Remarks A name is assigned to a device during creation.

See Also KMCreateController, KMCreateSercosAxis, KMCreateSerialAxis

Example See Example #4

KMGetDeviceProductClass

Get the product class for a device.

For C/C++:

KMProductClass KMGetDeviceProductClass(KMDevice device)

For Visual Basic:

KMGetDeviceProductClass(ByVal device As Long) As Integer

For C#:
System.Int32 IKMGetDeviceProductClass (System.IntPtr device)

device	Handle to the device
Return Value	The product class of the device.
Remarks	See the product class description. Product class is assigned by the API during device creation.
See Also	KMCreateController, KMCreateSercosAxis, KMCreateSerialAxis
Example	See Example #4

KMGetDeviceProductType

Get the product type for a device.

For C/C++:

KMProductType KMGetDeviceProductType(KMDevice device)

For Visual Basic:

KMGetDeviceProductType(ByVal device As Long) As Integer

For C#:
System.Int32 IKMGetDeviceProductType (System.IntPtr device)

device	Handle to the device
Return Value	The product type of the device.
Remarks	See product type description. Product type is assigned by the user during device creation.
See Also	KMCreateController, KMCreateSercosAxis, KMCreateSerialAxis
Example	See Example #4

KMGetDeviceCommType

Get the communication type for a device.

For C/C++:

KMCommType KMGetDeviceCommType(KMDevice device)

For Visual Basic:

KMGetDeviceCommType(ByVal device As Long) As Integer

For C#:
System.Int32 IKMGetDeviceCommType (System.IntPtr device)

device	Handle to the device
Return Value	The communication type of the device. See communication type description.
Remarks	Communication type is assigned during device creation depending on what KMCreate call is used.
See Also	KMCreateController, KMCreateSercosAxis, KMCreateSerialAxis
Example	See Example #4

KMGetDeviceCommPort

Get the COM port the device is attached to.

For C/C++:

KMErrorCode KMGetDeviceCommPort(KMDevice device, LPSTR commPort, long commPortLen)

For Visual Basic:

KMGetDeviceCommPort(ByVal device As Long, ByVal commPort$, ByVal commPortLen As Long) As Long

For C#:

System.Int32 IKMGetDeviceCommPort (System.IntPtr device , System.String commPort , System.UInt32 commPortLen)

device	Handle to the device
comPort	String with COM port that the device is connected to
comPortLen	Length of *comPort* string

Return Value KM_ERR_OK if command is successfully transmitted

Remarks Comm port is assigned during device creation.

Will return a blank (NULL) string if no COMM port assigned.

See Also KMCreateController, KMCreateSerialAxis

Example See Example #4

KMGetControllerID

Get the controller address for the controller.

For C/C++:

short KMGetControllerID(KMDevice controller)

For Visual Basic:

KMGetControllerID(ByVal controller As Long) As Integer

For C#:
System.Int32 IKMGetControllerID (System.IntPtr controller)

controller	Handle to the controller
Return Value	0 is returned if there is no controller ID for the device.
Remarks	ControllerID is assigned during controller creation.
See Also	KMCreateController
Example	See Example #4

Example #4

This program shows how to use the functions that access the device table.

C / C++

```
/* Declare device handles */
KMDevice devController1, devController2;
KMDevice axisA1;

/* Declare variables */
char buf[500];
KMDevice handle;

/* Initialize the API */
KMInitialize();

/* Create a plugin controller */
devController1 = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "" );

/* Create a controller on a serial port */
devController2 = KMCreateController( PROD_SSMC, "Secondary
controller", COMM_SERIAL, 2, "COM1" );

/* Create an axis on the main controller */
axisA1 = KMCreateSercosAxis( PROD_SERCOSTAR, “A1” );

/* What controller is the axis associated with? */
handle = KMGetAxisController( axisA1 );
if (KMGetDeviceProductClass(handle) == CLASS_CONTROLLER &&
KMGetDeviceProductType(handle) == PROD_SSMC )
{
   KMGetDeviceName( handle, buf, sizeof(buf) );
   printf(“The axis A1 is located on the %s.\n”, buf );
}
else
   printf(“Unexpected device type or class.\n”);

/* What Comm port is the secondary controller on? */
printf( “The secondary controller is communicating on %s.\n”,
KMGetDeviceCommType(devController2) );

/* What ID# is the secondary controller? */
printf( “The secondary controller is ID# %d.\n”, KMGetControllerID(
devController2 ) );

/* Destroy device table entries */
KMDestroyDevice( axisA1 );
KMDestroyDevice( devController1 );
KMDestroyDevice( devController2 );

/* Terminate the API */
KMTerminate();
```

Visual Basic

```
'Declare device handles
Dim devController1 As Long
Dim devController2 As Long
Dim axisA1 As Long

'Declare variables
buf$
Dim handle As Long
Dim errorL As Long

'Initialize the API
errorL = KMInitialize()

'Create a plugin controller
devController1 = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "" )

'Create a controller on a serial port
devController2 = KMCreateController( PROD_SSMC, "Secondary
controller", COMM_SERIAL, 2, "COM1" )

'Create an axis on the main controller
axisA1 = KMCreateSercosAxis( PROD_SERCOSTAR, “A1” )

'What controller is the axis associated with?
handle = KMGetAxisController( axisA1 )
If KMGetDeviceProductClass(handle) == CLASS_CONTROLLER &&
KMGetDeviceProductType(handle) == PROD_SSMC Then

   errorL = KMGetDeviceName( handle, buf, Length(buf) )
   Print "The axis A1 is located on the"; buf

Else
   Print "Unexpected device type or class."
EndIf

'What Comm port is the secondary controller on?
Dim commType As Long
commType = KMGetDeviceCommType(devController2)
Print "The secondary controller is communicating on"; commType

'What ID# is the secondary controller?
Dim contID As Long
contID = KMGetControllerID(devController2)
Print "The secondary controller is ID# "; contID

'Destroy device table entries
errorL = KMDestroyDevice( axisA1 )
errorL = KMDestroyDevice( devController1 )
errorL = KMDestroyDevice( devController2 )

'Terminate the API
errorL = KMTerminate()
```

Device Table Saving and Loading

KMWriteDeviceFile

Write present API configuration to a file.

For C/C++:

KMErrorCode KMWriteDeviceFile(LPSTR fileName)

For Visual Basic:

For C#:
System.Int32 IKMWriteDeviceFile (System.String fileName)

fileName	String with filename
Return Value	KM_ERR_OK if command is successfully
Remarks	API must have rights to write to *fileName*.
See Also	KMReadDeviceFile
Example	See Example #5

KMReadDeviceFile

Read a new API configuration in from a file.

KMErrorCode	KMReadDeviceFile(LPSTR fileName)
filename	String with filename
Return Value	KM_ERR_OK if command is successfully
Remarks	API must have rights to read from fileName.
See Also	KMWriteDeviceFile
Example	See Example #5

Example #5

This program shows how to read and write device tables.

C / C++

```
/* Initialize the API */
KMInitialize();

/* Read in device table */
KMReadDeviceFile( “API1.TBL” );

/* Write device table out to another file */
KMWriteDeviceFile( “API2.TBL” );

/* Terminate the API */
KMTerminate();
```

Visual Basic

```
'Create An Error Variable
Dim errorL As Long

'Initialize the API
errorL = KMInitialize()

'Read in device table
errorL = KMReadDeviceFile( “API1.TBL” )

'Write device table out to another file
erroL = KMWriteDeviceFile( “API2.TBL” )

'Terminate the API
errorL = KMTerminate()
```

Group Access Functions

KMCreateControllerGroup

Create a device table entry for a group.

For C/C++:

KMDevice KMCreateControllerGroup(LPSTR groupName, KMDevice controllerDevice)

For Visual Basic:

KMCreateControllerGroup(ByVal groupName$, ByVal controllerDevice As Long) As Long

For C#:
System.IntPtr IKMCreateControllerGroup (System.String groupName , System.IntPtr controllerDevice)

groupName	String name of the group
controllerDevice	Handle for the controller the group is on
Return Value	Handle to the group created or NULL on failure
Remarks	The *groupName* parameter must correspond with the name of the group on the controller.
See Also	KMCreateController, KMDestroyDevice
Example	See Example #6

KMAddAxisToGroup

Add an axis to a group list.

For C/C++:

KMErrorCode KMAddAxisToGroup(KMDevice group, KMDevice axis)

For Visual Basic:

KMAddAxisToGroup(ByVal group As Long, ByVal axis As Long) As Long

For C#:
System.Int32 IKMAddAxisToGroup (System.IntPtr group , System.IntPtr axis)

group	Handle of the group to add the axis to
axis	Handle of the axis being added
Return Value	KM_ERR_OK if command is successfully transmitted
Remarks	This function only updates the group list within the API. If the controller needs to be updated as well the user must do so explicitly.
See Also	KMCreateGroup, KMRemoveAxisFromGroup
Example	See Example #6

KMRemoveAxisFromGroup

Remove an axis from a group list.

For C/C++:

KMErrorCode KMRemoveAxisFromGroup(KMDevice group, KMDevice axis)

For Visual Basic:

KMRemoveAxisFromGroup(ByVal group As Long, ByVal axis As Long) As Long

For C#:
System.Int32 IKMRemoveAxisFromGroup (System.IntPtr group , System.IntPtr axis)

group	Handle of the group to remove the axis from
axis	Handle of the zxis to remove from the group

Return Value KM_ERR_OK if command is successfully transmitted

Remarks This function only updates the group list within the API. If the controller needs to be updated as well the user must do so explicitly.

When a group is destroyed the list of axes associated with that group is automatically destroyed as well. Therefore the user does not have to explicitly remove each axis from a group before program termination.

See Also KMCreateGroup, KMAddAxisToGroup

Example See Example #6

Example #6

This program shows how to use create groups entries in the device table.

C / C++

```
/* Allocate device handles */
KMDevice axisA1, axisA2;
KMDevice groupG1;
KMDevice devController;

/* Initialize the API */
KMInitialize();

/* Create a controller */
devController = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "" );

/* Create an axes on the main controller */
axisA1 = KMCreateSercosAxis( PROD_SERCOSTAR, “A1” );
axisA2 = KMCreateSercosAxis( PROD_SERCOSTAR, “A2” );

/* Create a group on the main controller */
groupG1 = KMCreateGroup( “G1”, devController);

/* Associate A1 and A2 with G1 */
KMAddAxisToGroup( G1, A1 );
KMAddAxisToGroup( G1, A2 );

/* Set VCRUISE to 1000 for G1 */
KMVariableSetLongValue(G1, ”VCRUISE”, 1000);

/* Destroy device table entries */
KMDestroyDevice( groupG1 );
KMDestroyDevice( axisA1 );
KMDestroyDevice( axisA2 );
KMDestroyDevice( devController );

/* Terminate the API */
KMTerminate();
```

Visual Basic

```
'Allocate device handles
Dim axisA1 As Long
Dim axisA2 As Long
Dim groupG1 As Long
Dim devController As Long

'Declare An Error Variable
Dim errorL As Long

'Initialize the API
errorL = KMInitialize()

'Create a controller
devController = KMCreateController( PROD_SSMC, "Main controller",
COMM_PLUGIN, 1, "" )

'Create an axes on the main controller
axisA1 = KMCreateSercosAxis( PROD_SERCOSTAR, "A1" )
axisA2 = KMCreateSercosAxis( PROD_SERCOSTAR, "A2" )

'Create a group on the main controller
groupG1 = KMCreateGroup( "G1", devController)

'Associate A1 and A2 with G1
errorL = KMAddAxisToGroup( G1, A1 )
errorL = KMAddAxisToGroup( G1, A2 )

'Set VCRUISE to 1000 for G1
errorL = KMVariableSetLongValue(G1, "VCRUISE", 1000)

'Destroy device table entries
errorL = KMDestroyDevice( groupG1 )
errorL = KMDestroyDevice( axisA1 )
errorL = KMDestroyDevice( axisA2 )
errorL = KMDestroyDevice( devController )

'Terminate the API
errorL = KMTerminate()
```

Device Table Iterators

KMCreateDeviceIterator

Create an iterator to traverse the device table.

For C/C++:

KMDeviceIterator KMCreateDeviceIterator(KMProductClass prodClass, KMDevice device)

For Visual Basic:

KMCreateDeviceIterator(ByVal prodClass As Integer, ByVal device As Long) As Long

For C#:
System.IntPtr IKMCreateDeviceIterator (System.UInt32 prodClass , System.IntPtr device)

prodClass	Product class to select an iterator for
device	Device to select an iterator for

Return Value Handle to the device iterator, NULL on failure

Remarks These combinations of prodClass and device type result in the following:

prodClass	device	result
CLASS_NONE	NULL	all controllers, groups and axes
CLASS_CONTROLLER	NULL	all controllers
CLASS_GROUP	NULL	all groups
CLASS_AXIS	NULL	all axes
CLASS_GROUP	CONTROLLER	all groups for a controller
CLASS_AXIS	CONTROLLER	all axes for a controller
CLASS_AXIS	GROUP	all axes for a group

All other combinations are invalid.

See Also KMDestroyDeviceIterator, KMGetNextDevice, KMGetPrevDevice, KMReadDeviceFile, KMWriteDeviceFile, KMCreateController, KMCreateSerialAxis, KMCreateSercosAxis

Example See Example #7

KMDestroyDeviceIterator

Destroy the device table iterator.

For C/C++:

KMErrorCode KMDestroyDeviceIterator(KMDeviceIterator deviceIter)

For Visual Basic:

KMDestroyDeviceIterator(ByVal deviceIter As Long) As Long

For C#:
System.Int32 IKMDestroyDeviceIterator (System.IntPtr deviceIter)

deviceIter	Handle for the iterator to destroy
Return Value	KM_ERR_OK if command is successfully transmitted
Remarks	Must be called to release memory allocated by the API for the user.
See Also	KMCreateDeviceIterator, KMGetNextDevice, KMGetPrevDevice
Example	See Example #7

KMGetNextDevice

Retrieve the next device in the device table.

For C/C++:

KMDevice KMGetNextDevice(KMDeviceIterator deviceIter)

For Visual Basic:

KMGetNextDevice(ByVal deviceIter As Long) As Long

For C#:
System.IntPtr IKMGetNextDevice (System.IntPtr deviceIter)

Return Value	Handle to the next device or NULL if another device is not found.
Remarks	Must create a device iterator first with KMCreateDeviceIterator.
See Also	KMCreateDeviceIterator, KMDestroyDeviceIterator, KMGetPrevDevice
Example	See Example #7

KMGetPrevDevice

Retrieve the previous device in the device table.

For C/C++:

KMDevice KMGetPrevDevice(KMDeviceIterator deviceIter)

For Visual Basic:

KMGetPrevDevice(ByVal deviceIter As Long) As Long

For C#:
System.Int32 IKMGetPrevDevice (System.IntPtr deviceIter)

Return Value	Handle to the previous device or NULL if another device is not found.
Remarks	Must create a device iterator first with KMCreateDeviceIterator.
See Also	KMCreateDeviceIterator, KMDestroyDeviceIterator, KMGetNextDevice
Example	See Example #7

Example #7

This program shows how to iterate the device table.

```
/* Initialize the API */
KMInitialize();

/* This example not complete. Contact factory for more information */

/* Terminate the API */
KMTerminate();
```

Asynchronous Message Handler

Asynchronous messages are messages that originate on a device and are not related to a command or action initiated through the API (i.e., they are not the response to a command). The API converts these messages into Windows messages. By default, the API then displays the message in a modal dialog box in the center of the user's screen. If the user wants to handle the asynchronous messages on their own they must handle the **WM_KM_ASYNC** message. In C/C++ and MFC this is straightforward, Visual Basic requires advanced programming (see "Visual Basic 5.0 Programmer's Guide to the Win32 API" by Dan Appleman) or the use of an ActiveX control such as SpyWorks by DesaWare.

If the user just wishes to disable display of the asynchronous messages this can be accomplished through the KMAsyncEnableMessages API.

See "Advanced asynchronous message handling" on page 4 also.

KMAsyncGetHandler

Returns the present asynchronous message handler.

For C/C++:

KMAsyncHandler KMAsyncGetHandler(void)

For Visual Basic:

KMAsyncGetHandler() As Long

For C#:
IKMAsyncGetHandler ( )

Return Value	Window handle (HWND) of present asynchronous message handler.
Remarks	Save return value before calling KMAsyncSetHandler.
See Also	KMAsyncGetHandler, KMAsyncSetHandler, KMAsyncGetMessage, KMAsyncEnableMessages
Example	See Example #8

KMAsyncSetHandler

Set the asynchronous message handler.

For C/C++:

KMErrorCode KMAsyncSetHandler(KMAsyncHandler hAsyncHandler)

For Visual Basic:

KMAsyncSetHandler(ByVal hAsyncHandler As Long) As Long

For C#:
System.Int32 IKMAsyncSetHandler (System.IntPtr hAsyncHandler)

hAsyncHandler	Window handle (HWND) to send asynchronous messages to
Return Value	KM_ERR_OK if command is successfully transmitted
Remarks	Call KMAsyncGetHandler and save the return value before calling KMAsyncSetHandler in order to restore prior value.
See Also	KMAsyncGetHandler, KMAsyncSetHandler, KMAsyncGetMessage, KMAsyncEnableMessages
Example	See Example #8

KMAsyncGetMessage

Gets the message contents and message type for the asynchronous message associated with the asynchronous message handle stored in WM_KM_ASYNC's lPARAM.

For C/C++:

KMErrorCode KMAsyncGetMessage(LPARAM hAsyncMsg, LPSTR lpszMessage, UINT nMessageSize, LPSTR nMessageType)

For Visual Basic:

KMAsyncGetMessage (ByVal hAsyncMsg As Long, ByVal lpszMessage$, ByVal nMessageSize As Long, ByVal nMessageType$) As Long

For C#:

System.Int32 IKMAsyncGetMessage (System.UInt32 hAsyncMsg , char[] lpszMessage , System.UInt32 nMessageSize , char[] nMessageType)

hAsyncMsg	Handle associated with the message (LPARAM).
lpszMessage	The text associated with the message.
nMessageSize	Size of *lpszMessage* buffer.
nMessageType	What type of message was received.

Return Value	KM_ERR_OK if message was retrieved successfully.
Remarks	This function can only be called once for each asynchronous message handled.
See Also	KMAsyncGetHandler, KMAsyncSetHandler, KMAsyncGetMessage, KMAsyncEnableMessages
Example	See Example #8

KMAsyncEnableMessages

Enables and disables the display of asynchronous messages both for the default display by the API and the message delivery to the user's application.

For C/C++:

KMErrorCode KMAsyncEnableMessages(UINT16 bVal)

For Visual Basic:

KMAsyncEnableMessages (ByVal bVal As Integer) As Long

For C#:
System.Int32 IKMAsyncEnableMessages (System.UInt16 bVal)

bVal	Non zero means display the asynchronous messages.
Return Value	KM_ERR_OK if function was executed successfully.
Remarks	
See Also	KMAsyncGetHandler, KMAsyncSetHandler, KMAsyncGetMessage, KMAsyncEnableMessages
Example	See Example #8

Example #8

This program shows how to set the async message handler.

```
/* Initialize the API */
KMInitialize();

/* Example to be completed. Contact factory for more information */

/* Terminate the API */
KMTerminate();
```

ServoStar MC Specific Functions

KMMCSetVin

Sets the value of the MC variable SYS.VIN via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCSetVin(KMDevice device, UINT32 nValue)

For Visual Basic:

KMMCSetVin (ByVal device As Long, ByVal dwValue As Long) As Long

For C#:
System.Int32 IKMMCSetVin (System.IntPtr device , System.UInt32 nValue)

device	Handle for the MC being accessed.
nValue	Value to set the virtual input to.

Return Value KM_ERR_OK on success

Remarks

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCGetVin

Gets the value of the MC variable SYS.VIN via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCGetVin(KMDevice device, LPUINT32 pnValue)

For Visual Basic:

KMMCGetVin (ByVal device As Long, lpdwValue As Long) As Long

For C#:
System.Int32 IKMMCGetVin (System.IntPtr device , System.UInt32 pnValue)

device	Handle for the MC being accessed.
pnValue	Pointer to a double word to store virtual input.

Return Value KM_ERR_OK on success

Remarks

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCGetVout

Gets the value of the MC variable SYS.VOUT via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCGetVout(KMDevice device, LPUINT32 pnValue)

For Visual Basic:

KMMCGetVout (ByVal device As Long, lpdwValue As Long) As Long

For C#:
System.Int32 IKMMCGetVout (System.IntPtr device , System.UInt32 pnValue)

device	Handle for the MC being accessed.
pnValue	Pointer to a double word to store virtual output.

Return Value KM_ERR_OK on success

Remarks

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCGetDin

Gets the value of the MC variable SYS.DIN via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCGetDin(KMDevice device, LPUINT32 pnValue)

For Visual Basic:

KMMCGetDin (ByVal device As Long, pnValue As Long) As Long

For C#:
System.Int32 IKMMCGetDin (System.IntPtr device , System.UInt32 pnValue)

device	Handle for the MC being accessed.
pnValue	Pointer to a double word to store digital input.

Return Value KM_ERR_OK on success

Remarks

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCGetDout

Gets the value of the MC variable SYS.DOUT via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCGetDout(KMDevice device, LPUINT32 pnValue)

For Visual Basic:

KMMCGetDout (ByVal device As Long, lpdwValue As Long) As Long

For C#:
System.Int32 IKMMCGetDout (System.IntPtr device , System.UInt32 pnValue)

device	Handle for the MC being accessed.
pnValue	Pointer to a double word to store digital output.

Return Value KM_ERR_OK on success

Remarks

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCGetMCDouble

Gets the value of the MC variable SYS.MCDOUBLE[] via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCGetMCDouble(KMDevice device, WORD nIndex, LPDOUBLE pfValue)

For Visual Basic:

KMMCGetMCDouble (ByVal device As Long, ByVal nIndex As Integer, lpfValue As Double) As Long

For C#:
System.Int32 IKMMCGetMCDouble (System.IntPtr device , System.UInt32 nIndex , System.Double pfValue)

device	Handle for the MC being accessed.
nIndex	Index of value to retrieve.
pfValue	Pointer to a double to store MC double.

Return Value	KM_ERR_OK on success
Remarks	Valid values of nIndex are 1-8.
See Also	KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData
Example	See Example #9

KMMCGetMCInteger

Gets the value of the MC variable SYS.MCINTEGER[] via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCGetMCInteger(KMDevice device, WORD nIndex, LPINT32 pnValue)

For Visual Basic:

KMMCGetMCInteger (ByVal device As Long, ByVal nIndex As Integer, pnValue As Long) As Long

For C#:
System.Int32 IKMMCGetMCInteger (System.IntPtr device , System.UInt32 nIndex , System.UInt32 pnValue)

device	Handle for the MC being accessed.
nIndex	Index of value to retrieve.
pnValue	Pointer to a double word to store MC integer.

Return Value KM_ERR_OK on success

Remarks Valid values of nIndex are 1-8.

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCSetHostDouble

Sets the value of the MC variable SYS.HOSTDOUBLE[] via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCSetHostDouble(KMDevice device, WORD nIndex, DOUBLE fValue)

For Visual Basic:

KMMCSetHostDouble (ByVal device As Long, ByVal nIndex As Integer, ByVal fValue As Double) As Long

For C#:
System.Int32 IKMMCSetHostDouble (System.IntPtr device , System.UInt32 nIndex , System.Double fValue)

device	Handle for the MC being accessed.
nIndex	Index of value to retrieve.
fValue	Value to set host double to.

Return Value KM_ERR_OK on success

Remarks Valid values of nIndex are 1-8.

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCSetHostInteger

Sets the value of the MC variable SYS.HOSTINTEGER[] via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCSetHostInteger(KMDevice device, WORD nIndex, INT32 nValue)

For Visual Basic:

KMMCSetHostInteger (ByVal device As Long, ByVal nIndex As Integer, ByVal dwValue As Long) As Long

For C#:

System.Int32 IKMMCSetHostInteger (System.IntPtr device , System.UInt32 nIndex , System.UInt32 nValue)

device	Handle for the MC being accessed.
nIndex	Index of value to retrieve.
nValue	Value to set host integer to.

Return Value KM_ERR_OK on success

Remarks Valid values of nIndex are 1-8.

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCGetHostDouble

Gets the value of the MC variable SYS.HOSTDOUBLE[] via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCGetHostDouble(KMDevice device, WORD nIndex, LPDOUBLE pfValue)

For Visual Basic:

KMMCGetHostDouble (ByVal Axis As Long, ByVal nIndex As Integer, pfValue As Double) As Long

For C#:

System.Int32 IKMMCGetHostDouble (System.IntPtr Axis , System.UInt32 nIndex , System.Double pfValue)

device	Handle for the MC being accessed.
nIndex	Index of value to retrieve.
pfValue	Pointer to a double store host double.

Return Value KM_ERR_OK on success

Remarks Valid values of nIndex are 1-8.

See Also KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCGetHostInteger

Gets the value of the MC variable SYS.HOSTINTEGER[] via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCSetHostInteger(KMDevice device, WORD nIndex, lpINT32 nValue)

For Visual Basic:

KMMCGetHostInteger (ByVal Axis As Long, ByVal nIndex As Integer, nValue As Long) As Long

For C#:

System.Int32 IKMMCGetHostInteger (System.IntPtr Axis , System.UInt32 nIndex , System.UInt32 nValue)

device	Handle for the MC being accessed.
nIndex	Index of value to retrieve.
nValue	Value to set host integer to.

Return Value	KM_ERR_OK on success
Remarks	Valid values of nIndex are 1-8.
See Also	KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble, KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData
Example	See Example #9

KMMCGetAxisData

Gets the position feedback, velocity feedback and status of axis on MC via the fast data transfer mechanism.

For C/C++:

KMErrorCode KMMCGetAxisData(KMDevice device, WORD nAxis, LPDOUBLE pfPfb, LPDOUBLE pfVfb, LPUINT16 lpnStatus)

For Visual Basic:

KMMCGetAxisData (ByVal device As Long, ByVal nAxis As Integer, lpfPfb As Double, lpfVfb As Double, lpnStatus As Integer) As Long

For C#:

System.Int32 IKMMCGetAxisData (System.IntPtr device , System.UInt32 nAxis, System.Double pfPfb, System.Double pfVfb, System.UInt32 lpnStatus)

device	Handle for the MC being accessed.
nAxis	Index of axis to get axis data from.
pfPfb	Pointer to a double to store axis position feedback.
pfVfb	Pointer to a double to store axis velocity feedback.
pnStatus	Pointer to a word to store axis status.

Return Value	KM_ERR_OK on success

Remarks

Valid values of nAxis are 1-8.

pnStatus is mapped as follows:

Bit 0 - IsSettled: Set to 1 when the generator has completed AND abs(Target_pos - Actual_pos) <= PositionErrorSettle

Bit 1 - IsMoving: Set to 1 when the profiler is active.

Bit 2 - DriveEnabled: Set to 1 when the drive associated with the axis is enabled.

Bit 3 - DriveFault: Set to 1 when a fault exists on the drive associated with the axis.

All unused bits return 0.

See Also	KMMCSetVin, KMMCGetVin, KMMCGetVout, KMMCGetDin, KMMCGetDout, KMMCGetMCDouble,

KMMCGetMCInteger, KMMCSetHostDouble, KMMCGetHostDouble, KMMCSetHostInteger, KMMCGetHostInteger, KMMCGetAxisData

Example See Example #9

KMMCGetNumControllers

Gets the number of controllers that are installed in the host computer.

For C/C++:

KMErrorCode KMMCGetNumControllers(LPINT16 lpnNumControllers)

For Visual Basic:

KMMCGetNumControllers(ByVal lpnNumControllers As Long) As Long

For C#:

System.Int32 IKMMCGetNumControllers (System.UInt32 lpnNumControllers)

lpnNumContrllers	Pointer to an integer to store the number of controllers.
Return Value	KM_ERR_OK on success
Remarks	None
See Also	None
Example	See Example #9

Example #9

This program shows how to use fast I/O on the ServoStar MC.

```
/* Initialize the API */
KMInitialize();

/* This example not complete. Contact factory for more information */

/* Terminate the API */
KMTerminate();
```

Serial and Ethernet Specific Functions

KMTCPRefreshDevices

Forces an update of the devices connected to the host computer via serial or Ethernet. This function uses the Kollmorgen RBOOTP protocol to automatically identify connected devices.

For C/C++:

KMErrorCode KMTCPRefreshDevices(void)

For Visual Basic:

KMTCPRefreshDevices() As Long

For C#:
System.Int32 IKMTCPRefreshDevices ( )

Return Value	KM_ERR_OK on success
Remarks	This function implements the Kollmorgen RBOOTP protocol.
See Also	KMTCPGetNumDevices, KMTCPGetDeviceInformation
Example	See Example #10

KMTCPGetNumDevices

Gets a count of the number of devices connected via TCP/IP (serial or Ethernet).

For C/C++:

KMErrorCode KMTCPGetNumDevices(LPUINT16 lpnNumDevices)

For Visual Basic:

KMTCPGetNumDevices(lpnNumDevices As Integer) As Long

For C#:

System.Int32 IKMTCPGetNumDevices (System.UInt32 lpnNumDevices)

lpnNumDevices	Pointer to a 16 bit integer to receive the number of devices
Return Value	KM_ERR_OK on success
Remarks	The count returned by **KMTCPGetNumDevices** is updated each time **KMTCPRefreshDevices** is called.
See Also	KMTCPRefreshDevices, KMTCPGetDeviceInformation
Example	See Example #10

KMTCPGetDeviceInformation

Gets the list of devices connected via TCP/IP (serial or Ethernet).

For C/C++:

KMErrorCode KMTCPGetDeviceInformation(LPSTR lpszNameArray[], LPSTR lpszAddressArray[], LPSTR lpszSNArray[], UINT16 DIPArray[], INT32 nNumDevices)

For Visual Basic:

KMTCPGetDeviceInformation(ByVal lpszNameArray As Long, ByVal lpszAddressArray As Long, ByVal lpszSNArray As Long, ByVal nNumDevices As Long) As Long

For C#:

System.Int32 IKMTCPGetDeviceInformation (System.IntPtr[] lpszNameArray , System.IntPtr[] lpszAddressArray , System.IntPtr[] lpszSNArray , System.IntPtr[] DIPArray , System.UInt32 nNumDevices)

lpszNameArray	an array of pointers to strings containing the names of the controllers
lpszAddressArray	an array of pointers to strings (at least 16 characters long) containing the IP addresses of the controllers
lpszSNArray	an array of pointers to strings containing the serial numbers of the controllers
DIPArray	an array of short integers (16-bit) containing the DIP switch addresses of the controllers
nNumDevices	the length (number of elements) of lpszNameArray, lpszAddressArray and lpszSNArray

Return Value KM_ERR_OK on success

Remarks If a controller is removed from the network or for other reasons is not communicating properly, the IP address for the controller will be set to 0.0.0.0 instead of deleting the entry from the table.

The Visual Basic prototype is somewhat unusual due to the fact that the function passes arrays of strings. Example #10 shows how to use this function.

See Also KMTCPGetNumDevices, KMTCPRefreshDevices

Example See Example #10

Example #10

This program shows how to retrieve the devices that are connected via Ethernet or serial from the API:

C / C++

```
unsigned short NumberDevices = 0;

/* Force the API to look for devices on the network */
KMTCPRefreshDevices();

/* Get the number of devices found on serial and/or Ethernet */
KMTCPGetNumDevices(&NumberDevices);

/* Initialize the array to retrieve all device names, */
/* serial numbers, and IP addresses */
char **NameArray= new char*[NumberDevices];
char **IPArray  = new char*[NumberDevices];
char **SNArray  = new char*[NumberDevices];
UINT16 *DIPArray = new UINT16[NumberDevices];

for(int j=0;j<NumberDevices;j++)
{
   NameArray[j]=new char[MAX_TCP_NM_LENGTH];
   IPArray[j]=new char[MAX_TCP_IP_LENGTH];
   SNArray[j]=new char[MAX_TCP_SN_LENGTH];
}

/* Retrieve the device information  */
KMTCPGetDeviceInformation(NameArray, IPArray, SNArray, DIPArray,
   NumberDevices);
```

Visual Basic

Appendix

Notes for Visual Basic Users

1. The Kollmorgen API is a set of DLLs. When DLLs return strings to VB they frequently appear to be corrupted, especially if there was data in the string before it was passed to the DLL function. This is due to the different methods of storing strings between VB and the method the DLLs use called "zero-terminated strings". Zero-terminate strings are commonly used in C/C++ programming. The end of the string is designated by a CHR$(0). In order to cleanup a string that has been returned by a DLL the user should scan the string for a CHR$(0) and then trim the characters to the right. Another approach to the problem is to make sure the string is clear before passing it to a DLL. This can be accomplished by assigning it to "vbNullString" prior to calling the DLL function.

Notes on Asynchronous Messages

1. Asynchronous messages are "posted" (PostMessage) by the API (as opposed to being "sent").
2. Asynchronous messages are sent to all applications that have called KMInitialize.

Glossary of Terms

API

Application Program Interface -- a library of functions which are provided for the user to simplify a task.

asynchronous messages

Messages which are generated by a device but are not in response to a particular command. For example warning and error messages (over temperature, over travel, etc.).

axis

A motor and amplifier. When a multiaxis controller is being used the axis also contains the portion of the controller related to the motor and amplifier.

controller

Typically a multiaxis controller. Commands the motor, through an amplifier, to move to various positions or velocity.

device

A device is how the way represents the physical objects (axis, group or controller) in your system.

device table

A database managed by the API which records enough information to communicate with the "device" specified by the user.

DLL

Dynamic Link Library

group

A collection of axes which are coordinated, usually by a multiaxis controller such as the ServoStar MC.

long

A common type of variable used in computer languages. In this case a 32-bit signed integer.

lParam

A double word (long) sized parameter for a Windows message.

RBOOTP

A Kollmorgen proprietary protocol, similar to the standard BOOTP protocol, which is used to identify devices connected to IP based networks (serial/PPP or Ethernet).

ServoStar

Family name for Kollmorgen's modern amplifier line.

ServoStar MC

Name of Kollmorgen's multiaxis motion controller.

string

A common type used in computer languages.

wParam

A word sized parameter for a Windows message.

Index