softMC

Servotronix Motion API

Reference Manual

Revision 1.0

SERVOTRONIX
always in motion™

Revision History

Document Revision	Date	Remarks
1.0	Nov. 2014	Initial release

Copyright Notice

Disclaimer

This product documentation was accurate and reliable at the time of its release. Servotronix Motion Control Ltd. reserves the right to change the specifications of the product described in this manual without notice at any time.

Trademarks

All marks in this manual are the property of their respective owners.

Contact Information

Servotronix Motion Control Ltd.
21C Yagia Kapayim Street
Petach Tikva 49130 Israel
Tel: +972 (3) 927 3800
Fax: +972 (3) 922 8075
Website: www.servotronix.com

Customer Service

Servotronix is committed to delivering quality customer service and support for all our products. Our goal is to provide our customers with the information and resources so that they are available, without delay, if and when they are needed. In order to serve in the most effective way, we recommend that you contact your local sales representative for order status and delivery information, product information and literature, and application and field technical assistance. If you are unable to contact your local sales representative for any reason, please use the most relevant of the contact details below:

For technical support, contact: tech.support@servotronix.com

To order products, contact: orders@servotronix.com

For all other inquiries regarding Servotronix products, contact: customer.service@servotronix.com

Contents

1 Introduction

1.1 Overview

The **Servotronix Motion API** (application programming interface) allows you to interface your machine controller with the **softMC** motion controller by providing the means for:

- Sending commands and receiving responses
- Reading and setting drive or controller variables
- Sending and retrieving files
- Error handling

The API provides a library of functions that allow you to describe your system in terms of devices (axes, groups, controllers) and then communicate with these devices individually.

Note: The term "SERCOS" appears in some function names due to legacy support. These functions can be applied to all types of axes, such as CAN or EtherCAT.

The Servotronix Motion API is available for Windows XP/Windows 7, and is compatible with a wide variety of 32/64-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.

The API is installed as part of the **ControlStudio** installation.

1.2 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.

1.3 Device Table

Central to the API is the device table, a database that is managed by the API. The device table is populated with devices by the programmer, and enables the execution of commands on any device in the device table. Most communication techniques are built-in to the API, allowing you to focus on high level programming instead of communication protocols.

After initializing the API, the next task is to describe your system to the API. A typical system contains a softMC motion controller with several connected **CDHD** servo drives. This process involves describing and adding the devices you want to control to the API library device table.

The API has a set of functions that allow you to add devices to the API device table. Each of the KMCreate functions returns a handle to the newly created device. Handles allow the API to know which device you are referring to.

The device table access functions can be used to retrieve a handle to a device or information about a particular device. For example, the function KMGetAxisController returns a handle to the controller for an associated axis.

Note: The handle returned by KMCreateSercosAxis is a handle to an axis. An axis encompasses data on 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.

Other functions perform actions such as retrieval of the name, ID, product type, communication type of a device.

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 on the iterator list.

To remove devices from the device table use the function KMDestroyDevice.

Device tables can be saved and loaded via the functions KMWriteDeviceFile and KMReadDeviceFile.

1.4 Accessing Devices

The API provides two mechanisms for communicating with devices: command execution and variable access.

Command Execution

Command execution is handled through two functions, KMExecuteCmd and KMExecuteCmdResponse.

Each of these functions takes a handle to the device on which the command should be executed and a string buffer containing 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 functions include functions to get and set variables for each type (long, double, string), as well as functions to access the two components of an axis (drive and controller).

The distinction between the different types of variables is for user convenience. You can access variables that are of type string on the softMC with the function KMVariableControllerGetStringValue.

1.5 Error Messages

Most of the API functions return a code that indicates the status 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.

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 that 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 that 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 originally sent by the device, can be retrieved via KMErrorGetOriginalDeviceMessage.

Error Codes

The following table lists the error codes that can be returned by the API.

Error Code	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

Error Code	Value (hex)
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

1.6 Termination (Cleanup)

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

1.7 Sending/Retrieving Files

For the sending and retrieval of files, the API provides the functions KMPutFile and KMGetFile. 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.

1.8 Advanced Asynchronous Message Handling

Controller-generated messages not related to a specific command are called asynchronous messages. Examples of asynchronous messages are over-speed warnings, servo drive faults such as limit switch closures, and runtime error messages from softMC tasks. When these messages are received, the API converts them into Windows messages (events) which the programmer can request to be delivered to their application.

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

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

Asynchronous messages are "posted" (PostMessage) by the API (as opposed to being "sent").

Asynchronous messages are sent to all applications that have called KMInitialize.

1.9 Programming with Visual Basic

The API is a set of DLLs. When DLLs return strings to Visual Basic 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 difference in the method of storing strings used 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 CHR$(0). To clean up a string that has been returned by a DLL, you need to scan the string for CHR$(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.

2 Functions

Note: The complete list of the **Return Values** that can be returned by the API appears in the *Error Codes* table.

2.1.1 API Initialization

KMInitialize

Prepares the API for use.

C/C++	KMErrorCode KMInitialize (void)
Visual Basic	KMInitialize() As Long
Return Value	KM_ERR_OK if API is successfully initialized
	KM_ERR_WINDOWS_API if the asynchronous message window cannot 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	Example #1

KMTerminate

Performs API cleanup required for proper application termination.

C/C++	KMErrorCode KMTerminate (void)
Visual Basic	KMTerminate() As Long
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	Example #1

2.2 Device Management

The product type, product class, and communication type constants, specified in the tables below, are applicable to the functions KMCreateController and KMCreateSERCOSAxis.

Product Type

The following constant is a valid product type:

Constant	Value
PROD_SSMC	3

Product Class

The following constants are valid product classes:

Constant	Value
CLASS_NONE	0
CLASS_AXIS	1
CLASS_GROUP	2
CLASS_CONTROLLER	3

Communication Type

The following constants are valid communication types:

Constant	Value	Remarks
COMM_NONE	0	
COMM_TCPIP or COMM_ETHERNET	5	Connect to IP address, with network scan
COMM_TCPIP_DEDICATED	7	Connect to fixed IP address, without network scan

KMCreateController

Creates a device table entry for a controller.

C/C++	KMDevice KMCreateController(KMProductType controllerProduct, LPSTR controllerName, KMCommType controllerCommType, short controllerID, LPSTR commOptions)
Visual Basic	KMCreateController (ByVal controllerProduct As Integer, ByVal controllerName$, ByVal controllerCommType As Integer, ByVal controllerID As Integer, ByVal commOptions$) As Long
controllerProduct	Type of controller to create.
controllerName	String name for the controller.

controllerCommType	How the API communicates with the controller.
controllerID	The controller's address.
commOptions	IP address and optional port, Controller name or a Serial Number
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 you 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.
	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 KMDestroyDevice
Example	Example #1

KMCreateSercosAxis

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

C/C++	KMDevice KMCreateSercosAxis(KMProductType axisProduct, LPSTR axisName, KMDevice controllerDevice)
Visual Basic	KMCreateSercosAxis(ByVal axisProduct As Integer, ByVal axisName$, ByVal controllerDevice As Long) As Long
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.
	Refer to tables above: Product Types, Product Classes, Communication Types
	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 KMDestroyDevice
Example	Example #1

KMDestroyDevice

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

C/C++	KMErrorCode KMDestroyDevice(KMDevice device)
Visual Basic	KMDestroyDevice(ByVal device As Long) As Long
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
Example	Example #1

2.3 Device Access

KMExecuteCmd

Sends a command to a device.

C/C++	KMErrorCode KMExecuteCmd(KMDevice device, LPSTR cmdStr)
Visual Basic	KMExecuteCmd(ByVal device As Long, ByVal cmdStr$) As Long
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	Example #1

KMExecuteCmdResponse

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

C/C++	KMErrorCode KMExecuteCmdResponse(KMDevice device, LPSTR cmdStr, LPSTR outStr, DWORD outSize)
Visual Basic	KMExecuteCmdResponse(ByVal device As Long, ByVal cmdStr$, ByVal outStr$, ByVal outSize As Long) As Long
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	Example #1

KMVariableControllerGetLongValue

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

C/C++	KMErrorCode KMVariableControllerGetLongValue (KMDevice device, LPSTR varName, LPLONG pValue)
Visual Basic	KMVariableControllerGetLongValue(ByVal device As Long, ByVal varName$, pValue As Long) As Long
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
Example	Example #1

KMVariableControllerGetDblValue

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

C/C++	KMErrorCode KMVariableControllerGetDoubleValue (KMDevice device, LPSTR varName, LPDOUBLE pValue)
Visual Basic	KMVariableControllerGetDblValue(ByVal device As Long, ByVal varName$, pValue As Double) As Long
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
Example	Example #1

KMVariableControllerGetStringValue

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

C/C++	KMErrorCode KMVariableControllerGetStringValue (KMDevice device, LPSTR varName, LPSTR pValue, long valueLen)
Visual Basic	KMVariableControllerGetStringValue(ByVal device As Long, ByVal varName$, ByVal pValue$, ByVal valueLen As Long) As Long
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
Example	Example #1

KMVariableControllerSetLongValue

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

C/C++	KMErrorCode KMVariableControllerSetLongValue (KMDevice device, LPSTR varName, long value)
Visual Basic	KMVariableControllerSetLongValue(ByVal device As Long, ByVal varName$, ByVal value As Long) As Long
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
Example	Example #1

KMVariableControllerSetDblValue

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

C/C++	KMErrorCode KMVariableControllerSetDblValue (KMDevice device, LPSTR varName, double value)
Visual Basic	KMVariableControllerSetDblValue(ByVal device As Long, ByVal varName$, ByVal value As Double) As Long
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
Example	Example #1

KMVariableControllerSetStringValue

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

C/C++	KMErrorCode KMVariableControllerSetStringValue (KMDevice device, LPSTR varName, LPSTR pValue)
Visual Basic	KMVariableControllerSetStringValue(ByVal device As Long, ByVal varName$, ByVal value$) As Long
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
Example	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_TCPIP, 1, "IP:90.0.0.1");
/* Create axes */
axisA1 = KMCreateSercosAxis("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_TCPIP, 1, "IP:90.0.0.1")
'Create axes
axisA1 = KMCreateSercosAxis(PROD_SSMC, "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.

C/C++	KMErrorCode KMGetFile(KMDevice device, LPSTR cmdStr, LPSTR filename)
Visual Basic	KMGetFile(ByVal device As Long, ByVal cmdStr$, ByVal filename$) As Long
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	Example #2

KMPutFile

Sends a file to a device.

C/C++	KMErrorCode KMPutFile(KMDevice device, LPSTR cmdStr, LPSTR filename)
Visual Basic	KMPutFile(ByVal device As Long, ByVal cmdStr$, ByVal filename$) As Long
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	Example #2

Example #2

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

C/C++

```
/* Declare device handles */
KMDevice devController;
/* Initialize the API */
KMInitialize();
/* Create a controller */
devController = KMCreateController(PROD_SSMC, "Main controller",
                                   COMM_TCPIP, 1, "IP:90.0.0.1");
/* 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_TCPIP, 1, "IP:90.0.0.1");
'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()
```

KMGetMaxControllerAxis

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

C/C++	short KMGetMaxControllerAxis(KMDevice controller);
Visual Basic	KMGetMaxControllerAxis(ByVal controller As Long) As Integer
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.

2.4 Error Handling

Error handling is one of the most important elements for creating a robust application. The 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:

- Internally, within the API: host computer runs out of memory
- Communication between the API and the device: device fails to respond in time
- Internally, within 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.

Note: The complete list of the **Return Values** that can be returned by the API appears in the *Error Codes* table.

Example #3

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
		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,
		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
```

KMErrorGetMessage

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

C/C++	KMErrorCode KMErrorGetMessage(KMErrorCode errCode, LPSTR buf, short bufLen)
Visual Basic	KMErrorGetMessage(ByVal errCode As Long, ByVal buf$, ByVal bufLen As Integer) As Long
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	Example #4

KMErrorGetDeviceMessage

Get original text and error number of last device error.

C/C++	KMErrorCode KMErrorGetDeviceMessage(KMErrorCodePtr pErrCode, LPSTR buf, short bufLen)
Visual Basic	KMErrorGetDeviceMessage(pErrCode As Long, ByVal buf$, ByVal bufLen As Integer) As Long
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	Example #4

KMErrorGetOriginalDeviceMessage

Retrieves entire original error message from buffer.

C/C++	KMErrorCode KMErrorGetOriginalDeviceMessage(LPSTR buf, short bufLen);
Visual Basic	KMErrorGetOriginalDeviceMessage(ByVal buf$, ByVal bufLen As Integer) As Long
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	Example #4

Example #4

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_TCPIP, 1, "IP:90.0.0.1");
/* 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_TCPIP, 1, "IP:90.0.0.1");
'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()
```

2.5 Device Table Access

KMGetAxisController

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

C/C++	KMDevice KMGetAxisController(KMDevice axis)
Visual Basic	KMGetAxisController(ByVal axis As Long) As Long
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 KMCreateController
Example	Example #5

KMGetDeviceName

Get the name assigned to the device.

C/C++	KMErrorCode KMGetDeviceName(KMDevice device, LPSTR name, long nameLen)
Visual Basic	KMGetDeviceName(ByVal device As Long, ByVal deviceName$, ByVal nameLen As Long) As Long
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
Example	Example #5

KMGetDeviceProductClass

Get the product class for a device.

C/C++	KMProductClass KMGetDeviceProductClass(KMDevice device)
Visual Basic	KMGetDeviceProductClass(ByVal device As Long) As Integer
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
Example	Example #5

KMGetDeviceProductType

Get the product type for a device.

C/C++	KMProductType KMGetDeviceProductType(KMDevice device)
Visual Basic	KMGetDeviceProductType(ByVal device As Long) As Integer
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
Example	v

KMGetDeviceCommType

Get the communication type for a device.

C/C++	KMCommType KMGetDeviceCommType(KMDevice device)
Visual Basic	KMGetDeviceCommType(ByVal device As Long) As Integer
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
Example	Example #5

KMGetDeviceCommPort

Get the COM port the device is attached to.

C/C++	KMErrorCode KMGetDeviceCommPort(KMDevice device, LPSTR commPort, long commPortLen)
Visual Basic	KMGetDeviceCommPort(ByVal device As Long, ByVal commPort$, ByVal commPortLen As Long) As Long
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
Example	Example #5

KMGetControllerID

Get the controller address for the controller.

C/C++	short KMGetControllerID(KMDevice controller)
Visual Basic	KMGetControllerID(ByVal controller As Long) As Integer
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	Example #5

Example #5

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_TCPIP, 1, "IP:90.0.0.1");
/* Create a controller on a serial port */
devController2 = KMCreateController(PROD_SSMC, "Secondary controller",
                                    COMM_TCPIP, 2, "IP:90.0.0.2");
/* Create an axis on the main controller */
axisA1 = KMCreateSercosAxis(COMM_TCPIP, “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_TCPIP, 1, "IP:90.0.0.1")
'Create a controller on a serial port
devController2 = KMCreateController(PROD_SSMC, "Secondary controller",
                                    COMM_TCPIP, 2, "IP:90.0.0.1")
'Create an axis on the main controller
axisA1 = KMCreateSercosAxis(PROD_SSMC, “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()
```

2.6 Device Table Saving and Loading

KMWriteDeviceFile

Write present API configuration to a file.

C/C++	KMErrorCode KMWriteDeviceFile(LPSTR fileName)
Visual Basic	
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	Example #6

KMReadDeviceFile

Read a new API configuration from a file.

C/C++	KMErrorCode KMReadDeviceFile(LPSTR fileName)
Visual Basic	
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	Example #6

Example #6

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
errroL = KMWriteDeviceFile("API2.TBL")
'Terminate the API
errorL = KMTerminate()
```

2.7 Device Table Iterators

KMCreateDeviceIterator

Create an iterator to traverse the device table.

C/C++	KMDeviceIterator KMCreateDeviceIterator(KMProductClass prodClass, KMDevice device)
Visual Basic	KMCreateDeviceIterator(ByVal prodClass As Integer, ByVal device As Long) As Long
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 KMCreateSercosAxis
Example	Example #7

KMDestroyDeviceIterator

Destroy the device table iterator.

C/C++	KMErrorCode KMDestroyDeviceIterator(KMDeviceIterator deviceIter)
Visual Basic	KMDestroyDeviceIterator(ByVal deviceIter As Long) As Long
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	Example #7

KMGetNextDevice

Retrieve the next device in the device table.

C/C++	KMDevice KMGetNextDevice(KMDeviceIterator deviceIter)
Visual Basic	KMGetNextDevice(ByVal deviceIter As Long) As Long
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	Example #7

KMGetPrevDevice

Retrieve the previous device in the device table.

C/C++	KMDevice KMGetPrevDevice(KMDeviceIterator deviceIter)
Visual Basic	KMGetPrevDevice(ByVal deviceIter As Long) As Long
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	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();
```

2.8 Asynchronous Message Handler

Asynchronous messages are messages that originate on a device. They are not related to a command or action initiated through the API; that is, 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 screen.

If you want to handle the asynchronous messages on your own, you must handle the WM_KM_ASYNC message. In C/C++ and MFC this is straightforward. Visual Basic requires advanced programming or the use of an ActiveX control.

Note: 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 received by the API. You can use KMAsyncGetMessage to get the asynchronous message from the API.

Disabling the display of the asynchronous messages can be accomplished through the function KMAsyncEnableMessages.

KMAsyncGetHandler

Returns the present asynchronous message handler.

C/C++	KMAsyncHandler KMAsyncGetHandler(void)
Visual Basic	KMAsyncGetHandler() As Long
Return Value	Window handle (HWND) of present asynchronous message handler.
Remarks	Save return value before calling KMAsyncSetHandler.
See Also	KMAsyncGetHandler KMAsyncSetHandler KMAsyncGetMessage KMAsyncEnableMessages
Example	Example #8

KMAsyncSetHandler

Set the asynchronous message handler.

C/C++	KMErrorCode KMAsyncSetHandler(KMAsyncHandler hAsyncHandler)
Visual Basic	KMAsyncSetHandler(ByVal hAsyncHandler As Long) As Long
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	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.

C/C++	KMErrorCode KMAsyncGetMessage(LPARAM hAsyncMsg, LPSTR lpszMessage, UINT nMessageSize, LPSTR nMessageType)
Visual Basic	KMAsyncGetMessage (ByVal hAsyncMsg As Long, ByVal lpszMessage$, ByVal nMessageSize As Long, ByVal nMessageType$) As Long
hAsyncMsg	Handle associated with the message (LPARAM).
lpszMessage	The text associated with the message.
nMessageSize	Size of *lpszMessage* buffer.
nMessageType	The type of message 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	Example #8

KMAsyncEnableMessages

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

C/C++	KMErrorCode KMAsyncEnableMessages(UINT16 bVal)
Visual Basic	KMAsyncEnableMessages (ByVal bVal As Integer) As Long
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	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();
```

2.9 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 RBOOTP protocol to automatically identify connected devices.

C/C++	KMErrorCode KMTCPRefreshDevices(void)
Visual Basic	KMTCPRefreshDevices() As Long
Return Value	KM_ERR_OK on success
Remarks	This function implements the RBOOTP protocol.

See Also	KMTCPGetNumDevices KMTCPGetDeviceInformation
Example	Example #9

KMTCPGetNumDevices

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

C/C++	KMErrorCode KMTCPGetNumDevices(LPUINT16 lpnNumDevices)
Visual Basic	KMTCPGetNumDevices(lpnNumDevices As Integer) As Long
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	Example #9

KMTCPGetDeviceInformation

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

C/C++	KMErrorCode KMTCPGetDeviceInformation(LPSTR lpszNameArray[], LPSTR lpszAddressArray[], LPSTR lpszSNArray[], UINT16 DIPArray[], INT32 nNumDevices)
Visual Basic	KMTCPGetDeviceInformation(ByVal lpszNameArray As Long, ByVal lpszAddressArray As Long, ByVal lpszSNArray As Long, ByVal nNumDevices As Long) As Long
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 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.
	Visual Basic is somewhat unusual in that the function passes arrays of strings. Example #9 shows how to use this function.

See Also	KMTCPGetNumDevices KMTCPRefreshDevices
Example	Example #9

Example #9

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

```
Private Sub Command1_Click()
'Declare variables
Dim NumberDevices As Integer
Dim NameString(100) As String
Dim NumberDevicesas As Integer
Dim IPString(100) As String
Dim DIPArray(100) As Integer
Dim SerialString(100) As String
Dim err As Long
Dim tempstr As String

'Initialize the API
errorL = KMInitialize()
‘refresh the list of tcp devices
err = KMTCPRefreshDevices()
' Get the number of devices found on serial and/or Ethernet
err = KMTCPGetNumDevices(NumberDevices)
' Initialize the array to retrieve all device names,
' serial numbers, and IP addresses
     For i = 0 To 100
```

```
            NameString(i) = Space(20)
            SerialString(i) = Space(10)
            IPString(i) = Space(20)
    Next i
' Retrieve the device information
err = KMTCPGetDeviceInformation(VarPtr(NameString(0)), VarPtr(IPString(0)),
VarPtr(SerialString(0)), VarPtr(DIPArray(0)), NumberDevices)
For i = 0 To NumberDevices - 1
    NameString(i) = StrConv(NameString(i), vbUnicode)
    IPString(i) = StrConv(IPString(i), vbUnicode)
    SerialString(i) = StrConv(SerialString(i), vbUnicode)
    tempstr = ClipString(NameString(i)) + " " + ClipString(IPString(i)) + " " +
    ClipString(SerialString(i))
    List1.AddItem (tempstr), i
Next
List1.Visible = True
'Terminate the API
errorL = KMTerminate()
End Sub

Function ClipString(InputString As String) As String
'This function trims the null characters off the right side of a long string
    Dim xx As Integer
    xx = InStr(1, InputString$, Chr$(0)) 'find the first null character
    If xx Then
        ClipString$ = Mid$(InputString$, 1, xx - 1)
    Else
        ClipString$ = InputString$
    End If
End Function
```

3 Glossary

API	Application programming interface: a set of routines and tools that simplify the development of software applications.
asynchronous messages	Messages generated by a device that are not in response to a particular command, such as warnings and error messages.
axis	A motor and drive. When a multi-axis controller is used, also includes the components of the controller related to the motor and drive.
controller	Commands the motor, through a drive, to move to various positions or at a velocity.
device	Represents the physical objects (axis, group or controller) in the system.
device table	A database managed by the API that enables communication with devices specified by the user.
DLL	Dynamic link library
group	A collection of axes that are coordinated, usually by a multi-axis controller such as the softMC.
long	A common type of variable used in computer languages. In this case a 32-bit signed integer.
lParam	A double word size (long) parameter for a Windows message.
RBOOTP	A proprietary protocol, similar to the standard BOOTP protocol, used to identify devices connected to IP-based networks (serial/PPP or Ethernet).
CDHD	Name of Servotronix servo drive product line.
softMC	Name of Servotronix multi-axis motion controller.
string	A common type used in computer languages.
wParam	A word size parameter for a Windows message.

Servotronix - 21C Yagia Kapayim St.
POB 3919 Petach Tikva 49130, Israel
Tel: 972-3-927-3800
info@servotronix.com
www.servotronix.com