

# **softMC**

## **Servotronix Motion API**

### **Reference Manual**

**Revision 1.0**





## Revision History

| Document Revision | Date      | Remarks         |
|-------------------|-----------|-----------------|
| 1.0               | Nov. 2014 | Initial release |

## Copyright Notice

© 2014 Servotronix Motion Control Ltd.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means without prior written permission of Servotronix Motion Control Ltd..

## 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](http://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](mailto:tech.support@servotronix.com)

To order products, contact: [orders@servotronix.com](mailto:orders@servotronix.com)

For all other inquiries regarding Servotronix products, contact: [customer.service@servotronix.com](mailto:customer.service@servotronix.com)



# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                    | <b>7</b>  |
| 1.1      | Overview                               | 7         |
| 1.2      | Initialization                         | 7         |
| 1.3      | Device Table                           | 7         |
| 1.4      | Accessing Devices                      | 8         |
|          | Command Execution                      | 8         |
|          | Variable Access                        | 8         |
| 1.5      | Error Messages                         | 8         |
|          | Error Codes                            | 9         |
| 1.6      | Termination (Cleanup)                  | 10        |
| 1.7      | Sending/Retrieving Files               | 10        |
| 1.8      | Advanced Asynchronous Message Handling | 11        |
| 1.9      | Programming with Visual Basic          | 11        |
| <b>2</b> | <b>Functions</b>                       | <b>13</b> |
| 2.1.1    | API Initialization                     | 13        |
|          | KMInitialize                           | 13        |
|          | KMTerminate                            | 13        |
| 2.2      | Device Management                      | 14        |
|          | KMCreateController                     | 14        |
|          | KMCreateSercosAxis                     | 15        |
|          | KMDestroyDevice                        | 16        |
| 2.3      | Device Access                          | 16        |
|          | KMExecuteCmd                           | 16        |
|          | KMExecuteCmdResponse                   | 17        |
|          | KMVariableControllerGetLongValue       | 17        |
|          | KMVariableControllerGetDbIValue        | 18        |
|          | KMVariableControllerGetStringValue     | 18        |
|          | KMVariableControllerSetLongValue       | 19        |
|          | KMVariableControllerSetDbIValue        | 19        |
|          | KMVariableControllerSetStringValue     | 20        |
|          | Example #1                             | 20        |
|          | KMGetFile                              | 21        |
|          | KMPutFile                              | 22        |
|          | Example #2                             | 22        |
|          | KMGetMaxControllerAxis                 | 23        |
| 2.4      | Error Handling                         | 23        |
|          | Example #3                             | 24        |
|          | KMErrorGetMessage                      | 25        |
|          | KMErrorGetDeviceMessage                | 25        |
|          | KMErrorGetOriginalDeviceMessage        | 26        |
|          | Example #4                             | 26        |
| 2.5      | Device Table Access                    | 27        |
|          | KMGetAxisController                    | 27        |
|          | KMGetDeviceName                        | 28        |
|          | KMGetDeviceProductClass                | 28        |
|          | KMGetDeviceProductType                 | 28        |
|          | KMGetDeviceCommType                    | 29        |
|          | KMGetDeviceCommPort                    | 29        |
|          | KMGetControllerID                      | 29        |
|          | Example #5                             | 30        |
| 2.6      | Device Table Saving and Loading        | 31        |
|          | KMWriteDeviceFile                      | 31        |
|          | KMReadDeviceFile                       | 32        |

---

|  |           |
|--|-----------|
| Example #6 .....                                 | 32        |
| 2.7 Device Table Iterators .....                 | 32        |
| KMCreateDeviceIterator.....                      | 32        |
| KMDestroyDeviceIterator .....                    | 33        |
| KMGetNextDevice .....                            | 33        |
| KMGetPrevDevice .....                            | 34        |
| Example #7 .....                                 | 34        |
| 2.8 Asynchronous Message Handler .....           | 34        |
| KMAsyncGetHandler .....                          | 35        |
| KMAsyncSetHandler .....                          | 35        |
| KMAsyncGetMessage.....                           | 35        |
| KMAsyncEnableMessages.....                       | 36        |
| Example #8 .....                                 | 36        |
| 2.9 Serial and Ethernet Specific Functions ..... | 36        |
| KMTCPRefreshDevices .....                        | 36        |
| KMTCPGetNumDevices .....                         | 37        |
| KMTCPGetDeviceInformation .....                  | 37        |
| Example #9 .....                                 | 38        |
| <b>3 Glossary .....</b>                          | <b>41</b> |

---

# 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.

| <b>Error Code</b>                        | <b>Value (hex)</b>     |
|--|------------------------|
| <code>KM_ERR_OK</code>                   | <code>H00000000</code> |
| <code>KM_ERR_BAD</code>                  | <code>HFFFFFFF</code>  |
| <code>KM_ERR_BAD_CRC</code>              | <code>H00800001</code> |
| <code>KM_ERR_TIME_OUT</code>             | <code>H00800002</code> |
| <code>KM_ERR_BAD_DEVICE_ID</code>        | <code>H00800003</code> |
| <code>KM_ERR_DEVICE_NOT_INSTALLED</code> | <code>H00800004</code> |
| <code>KM_ERR_LOCK_FAILED</code>          | <code>H00800005</code> |
| <code>KM_ERR_NO_DATA</code>              | <code>H00800006</code> |
| <code>KM_ERR_BUFFER_TOO_SMALL</code>     | <code>H00800007</code> |
| <code>KM_ERR_ASYNC</code>                | <code>H00800008</code> |
| <code>KM_ERR_FAIL_SEND_MSG</code>        | <code>H00800009</code> |
| <code>KM_ERR_FAIL_ACK_NAK</code>         | <code>H0080000A</code> |
| <code>KM_ERR_FAIL_ACCEPT</code>          | <code>H0080000B</code> |
| <code>KM_ERR_FAIL_PROTOCOL</code>        | <code>H0080000C</code> |
| <code>KM_ERR_FAIL_RESPOND</code>         | <code>H0080000D</code> |
| <code>KM_ERR_UNKNOWN_MESSAGE</code>      | <code>H0080000E</code> |
| <code>KM_ERR_MEM_LOCK</code>             | <code>H0080000F</code> |
| <code>KM_ERR_FAIL_CREATE_FILE</code>     | <code>H00800010</code> |
| <code>KM_ERR_FAIL_OPEN_FILE</code>       | <code>H00800011</code> |
| <code>KM_ERR_FAIL_FIND_FILE</code>       | <code>H00800012</code> |
| <code>KM_ERR_UNEXPECTED_EOF</code>       | <code>H00800013</code> |
| <code>KM_ERR_NO_CONTEXT</code>           | <code>H00800014</code> |
| <code>KM_ERR_MEM_ALLOC</code>            | <code>H00800015</code> |
| <code>KM_ERR_INVALID_PRODUCT</code>      | <code>H00800016</code> |
| <code>KM_ERR_SYSTEM</code>               | <code>H00800017</code> |
| <code>KM_ERR_FAIL_FIND_NAME</code>       | <code>H00800018</code> |
| <code>KM_ERR_INVALID_TYPE</code>         | <code>H00800019</code> |
| <code>KM_ERR_NO_DEFAULT</code>           | <code>H0080001A</code> |

| <b>Error Code</b>             | <b>Value (hex)</b> |
|-------------------------------|--------------------|
| 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 by VB and the method the DLLs use called "zero-terminated strings". Zero-terminated 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.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMInitialize (void)  |
| <b>Visual Basic</b> | KMInitialize() As Long   |
| <b>Return Value</b> | KM_ERR_OK if API is successfully initialized<br>KM_ERR_WINDOWS_API if the asynchronous message window cannot be created.<br>KM_ERR_MEM_LOCK if the application context was not successfully created. |
| <b>Remarks</b>      | The KMInitialize function must be called before any other API functions are called.  |
| <b>See Also</b>     | KMTerminate  |
| <b>Example</b>      | <a href="#">Example #1</a>   |

#### KMTerminate

Performs API cleanup required for proper application termination.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMTerminate (void)   |
| <b>Visual Basic</b> | KMTerminate() As Long  |
| <b>Return Value</b> | KM_ERR_OK if API is successfully terminated.<br>KM_ERR_MEM_LOCK if the application context cannot be locked or removed.<br>KM_ERR_NO_CONTEXT if the application context cannot be found in the application context registry. |
| <b>Remarks</b>      | The KMTerminate function must be called before the application using the API terminates.<br>KMTerminate can be called at any time in order to flush the present device table.  |
| <b>See Also</b>     | KMInitialize   |
| <b>Example</b>      | <a href="#">Example #1</a>   |

## 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.

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

|                           |   |
|---------------------------|---|
| <i>controllerCommType</i> | How the API communicates with the controller.   |
| <i>controllerID</i>       | The controller's address.   |
| <i>commOptions</i>        | IP address and optional port, Controller name or a Serial Number  |
| <b>Return Value</b>       | A handle referring to the controller.   |
| <b>Remarks</b>            | <p>See product type description for <i>controllerProduct</i>.</p> <p><i>controllerName</i> is provided for user convenience and is not used internally by the API.</p> <p>See communication type description for <i>controllerCommType</i>.</p> <p><i>commOptions</i> 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 <i>commOptions</i> parameter allows you to specify what controller to connect to by IP address, controller name, serial number or DIP switch setting. The format for <i>commOptions</i> is as follows:</p> <ul style="list-style-type: none"> <li>■ "IP: xx.xx.xx.xx" where xx.xx.xx.xx is an IP address (e.g. "192.10.34.6").</li> <li>■ "NM: yyyy" where yyyy is a name assigned by the user to the Sys.Name property of the controller.</li> <li>■ "SN: XXXXX-XXX" where XXXXX-XXX is the serial number assigned to the controller at the factory and printed on the model number label.</li> </ul> <p>When connecting via serial the IP address should is always be specified as 91.0.0.2.</p> <p>If <i>commOptions</i> is not used, the parameter should be an empty string ("").</p> |
|                           | <p>KMCreateController allocates memory on behalf of the user. This memory must be freed before program termination via KMDestroyDevice.</p> <p>The range for <i>controllerDevice</i> is 1 - 9.</p>  |
| <b>See Also</b>           | KMCreateSercosAxis<br>KMDestroyDevice   |
| <b>Example</b>            | <a href="#">Example #1</a>  |

## KMCreateSercosAxis

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

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMDevice KMCreateSercosAxis(KMProductType axisProduct, LPSTR axisName, KMDevice controllerDevice)          |
| <b>Visual Basic</b> | KMCreateSercosAxis(ByVal axisProduct As Integer, ByVal axisName\$, ByVal controllerDevice As Long) As Long |
| <i>axisProduct</i>  | Type of axis to create   |

|                         |   |
|-------------------------|---|
| <i>axisName</i>         | String name for the axis  |
| <i>controllerDevice</i> | Handle of the controller the axis is attached to  |
| <b>Return Value</b>     | A handle referring to the axis.   |
| <b>Remarks</b>          | <p>KMCreateSercosAxis allocates memory on behalf of the user. This memory must be freed before program termination via KMDestroyDevice.</p> <p>Refer to tables above: Product Types, Product Classes, Communication Types</p> <p>See product type description for <i>axisProduct</i>.</p> <p>The API uses <i>axisName</i> to access variables on the axis, therefore it must match the name as it is set within the controller.</p> <p><i>controllerDevice</i> is the handle of the controller the axis is physically connected to.</p> |
| <b>See Also</b>         | KMCreateController<br>KMDestroyDevice   |
| <b>Example</b>          | <a href="#">Example #1</a>  |

## KMDestroyDevice

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

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMDestroyDevice(KMDevice device)                       |
| <b>Visual Basic</b> | KMDestroyDevice(ByVal device As Long) As Long                      |
| <i>device</i>       | Handle for the device to be destroyed.                             |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted.                  |
| <b>Remarks</b>      | The device handle will not be valid after calling KMDestroyDevice. |
| <b>See Also</b>     | KMCreateController<br>KMCreateSercosAxis                           |
| <b>Example</b>      | <a href="#">Example #1</a>   |

## 2.3 Device Access

### KMExecuteCmd

Sends a command to a device.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMExecuteCmd(KMDevice device, LPSTR cmdStr)    |
| <b>Visual Basic</b> | KMExecuteCmd(ByVal device As Long, ByVal cmdStr\$) As Long |
| <i>device</i>       | Handle for the device the command is being sent to.        |
| <i>cmdStr</i>       | The command to send.                                       |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted.          |



|                 |  |
|-----------------|--|
| <b>Remarks</b>  | Sends the command <i>cmdStr</i> unmodified to the device specified by <i>device</i> . Should be used with commands that do not have responses. |
| <b>See Also</b> | KMExecuteCmdResponse<br>KMErrorGetMessage<br>KMErrorGetDeviceMessage   |
| <b>Example</b>  | <a href="#">Example #1</a>   |

## KMExecuteCmdResponse

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

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrorCode KMExecuteCmdResponse(KMDevice device, LPSTR cmdStr, LPSTR outStr, DWORD outSize)  |
| <b>Visual Basic</b> | KMExecuteCmdResponse(ByVal device As Long, ByVal cmdStr\$, ByVal outStr\$, ByVal outSize As Long) As Long                               |
| <i>device</i>       | Handle for the device the command is being sent to.   |
| <i>cmdStr</i>       | The command to send.  |
| <i>outStr</i>       | A buffer for the response.  |
| <i>outSize</i>      | Size of the <i>outStr</i> buffer.   |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted  |
| <b>Remarks</b>      | Sends the command <i>cmdStr</i> unmodified to the device specified by <i>device</i> . Should be used with commands that have responses. |
| <b>See Also</b>     | KMExecuteCmd<br>KMErrorGetMessage<br>KMErrorGetDeviceMessage  |
| <b>Example</b>      | <a href="#">Example #1</a>  |

## KMVariableControllerGetLongValue

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

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrorCode KMVariableControllerGetLongValue (KMDevice device, LPSTR varName, LPLONG pValue)    |
| <b>Visual Basic</b> | KMVariableControllerGetLongValue(ByVal device As Long, ByVal varName\$, pValue As Long) As Long |
| <i>device</i>       | Handle for the device that contains the variable.   |
| <i>varName</i>      | Name of the variable.   |
| <i>pValue</i>       | Pointer to a long integer to receive the value of the device variable.                          |
| <b>Return Value</b> | KM_ERR_OK on success  |
| <b>Remarks</b>      | <i>varName</i> is the name of the variable as it is on the device.                              |

|                 |  |
|-----------------|--|
| <b>See Also</b> | KMVariableControllerGetDbIValue<br>KMVariableControllerGetStringValue<br>KMVariableControllerSetLongValue<br>KMVariableControllerSetDbIValue<br>KMVariableControllerSetStringValue |
| <b>Example</b>  | <a href="#">Example #1</a>   |

## KMVariableControllerGetDbIValue

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

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrorCode KMVariableControllerGetDoubleValue (KMDevice device, LPSTR varName, LPDOUBLE pValue)  |
| <b>Visual Basic</b> | KMVariableControllerGetDbIValue(ByVal device As Long, ByVal varName\$, pValue As Double) As Long  |
| <i>device</i>       | Handle for the device that contains the variable.   |
| <i>varName</i>      | Name of the variable.   |
| <i>pValue</i>       | Pointer to a double to receive the value of the device variable.  |
| <b>Return Value</b> | KM_ERR_OK on success  |
| <b>Remarks</b>      | varName is the name of the variable as it is on the device.   |
| <b>See Also</b>     | KMVariableControllerGetLongValue<br>KMVariableControllerGetStringValue<br>KMVariableControllerSetLongValue<br>KMVariableControllerSetDbIValue<br>KMVariableControllerSetStringValue |
| <b>Example</b>      | <a href="#">Example #1</a>  |

## KMVariableControllerGetStringValue

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

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrorCode KMVariableControllerGetStringValue (KMDevice device, LPSTR varName, LPSTR pValue, long valueLen)              |
| <b>Visual Basic</b> | KMVariableControllerGetStringValue(ByVal device As Long, ByVal varName\$, ByVal pValue\$, ByVal valueLen As Long) As Long |
| <i>device</i>       | Handle for the device that contains the variable.   |
| <i>varName</i>      | Name of the variable.   |
| <i>pValue</i>       | Pointer to a character buffer to receive the value of the device variable.  |
| <i>valueLen</i>     | Length of the character buffer <i>pValue</i> .  |
| <b>Return Value</b> | KM_ERR_OK on success  |
| <b>Remarks</b>      | varName is the name of the variable as it is on the device.   |

|                 |  |
|-----------------|--|
| <b>See Also</b> | KMVariableControllerGetLongValue<br>KMVariableControllerGetDbIValue<br>KMVariableControllerSetLongValue<br>KMVariableControllerSetDbIValue<br>KMVariableControllerSetStringValue |
| <b>Example</b>  | <a href="#">Example #1</a>   |

## KMVariableControllerSetLongValue

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

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMVariableControllerSetLongValue (KMDevice device, LPSTR varName, long value)  |
| <b>Visual Basic</b> | KMVariableControllerSetLongValue(ByVal device As Long, ByVal varName\$, ByVal value As Long) As Long   |
| <i>device</i>       | Handle for the device that contains the variable.  |
| <i>varName</i>      | Name of the variable.  |
| <i>value</i>        | Value to set the variable <i>varName</i> to.   |
| <b>Return Value</b> | KM_ERR_OK on success   |
| <b>Remarks</b>      | <i>varName</i> is the name of the variable as it is on the device.   |
| <b>See Also</b>     | KMVariableControllerGetLongValue<br>KMVariableControllerGetDbIValue<br>KMVariableControllerGetStringValue<br>KMVariableControllerSetDbIValue<br>KMVariableControllerSetStringValue |
| <b>Example</b>      | <a href="#">Example #1</a>   |

## KMVariableControllerSetDbIValue

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

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrorCode KMVariableControllerSetDbIValue (KMDevice device, LPSTR varName, double value)  |
| <b>Visual Basic</b> | KMVariableControllerSetDbIValue(ByVal device As Long, ByVal varName\$, ByVal value As Double) As Long   |
| <i>device</i>       | Handle for the device that contains the variable.   |
| <i>varName</i>      | Name of the variable.   |
| <i>value</i>        | Value to set the variable <i>varName</i> to.  |
| <b>Return Value</b> | KM_ERR_OK on success  |
| <b>Remarks</b>      | <i>varName</i> is the name of the variable as it is on the device.  |
| <b>See Also</b>     | KMVariableControllerGetLongValue<br>KMVariableControllerGetDbIValue<br>KMVariableControllerGetStringValue<br>KMVariableControllerSetLongValue<br>KMVariableControllerSetStringValue |
| <b>Example</b>      | <a href="#">Example #1</a>  |

## KMVariableControllerSetStringValue

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

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMVariableControllerSetStringValue (KMDevice device, LPSTR varName, LPSTR pValue)  |
| <b>Visual Basic</b> | KMVariableControllerSetStringValue(ByVal device As Long, ByVal varName\$, ByVal value\$) As Long   |
| <i>device</i>       | Handle for the device that contains the variable.  |
| <i>varName</i>      | Name of the variable.  |
| <i>pValue</i>       | Pointer to a character buffer to set the variable <i>varName</i> to.   |
| <b>Return Value</b> | KM_ERR_OK on success   |
| <b>Remarks</b>      | <i>varName</i> is the name of the variable as it is on the device.   |
| <b>See Also</b>     | KMVariableControllerGetLongValue<br>KMVariableControllerGetDbIValue<br>KMVariableControllerGetStringValue<br>KMVariableControllerSetLongValue<br>KMVariableControllerSetDbIValue |
| <b>Example</b>      | <a href="#">Example #1</a>   |

### 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);
KMVariableControllerGetDbIValue(axisA1, "PFINAL", &valueD);
KMVariableControllerGetStringValue(axisA1, "AMAX", valueS, sizeof(valueS));
/* Set variables */
KMVariableControllerSetLongValue(axisA1, "VCRUISE", valueL);
KMVariableControllerSetDbIValue(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 = KMVariableControllerGetDbIValue(axisA1, "PFINAL", valueD)
errorL = KMVariableControllerGetStringValue(axisA1, "AMAX", valueS,
                                             Length(valueS))
'Set variables
errorL = KMVariableControllerSetLongValue(axisA1, "VCRUISE", valueL)
errorL = KMVariableControllerSetDbIValue(axisA1, "PFINAL", valueD)
errorL = KMVariableControllerSetStringValue(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.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMGetFile(KMDevice device, LPSTR cmdStr, LPSTR filename)                                 |
| <b>Visual Basic</b> | KMGetFile(ByVal device As Long, ByVal cmdStr\$, ByVal filename\$) As Long                            |
| <i>device</i>       | Handle for the device the file is being retrieved from.  |
| <i>cmdStr</i>       | The command string to cause the device to send a file.   |
| <i>filename</i>     | Filename to save file to on host computer.   |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted   |
| <b>Remarks</b>      | The filename (if there is one) in <i>cmdStr</i> and <i>filename</i> parameters do not have to match. |

|                 |                            |
|-----------------|----------------------------|
| <b>See Also</b> | KMPutFile                  |
| <b>Example</b>  | <a href="#">Example #2</a> |

## KMPutFile

Sends a file to a device.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMPutFile(KMDevice device, LPSTR cmdStr, LPSTR filename)                                 |
| <b>Visual Basic</b> | KMPutFile(ByVal device As Long, ByVal cmdStr\$, ByVal filename\$) As Long                            |
| <i>device</i>       | Handle for the device the file is being sent to.   |
| <i>cmdStr</i>       | The command string to cause the device to retrieve a file.   |
| <i>filename</i>     | Filename of file on host computer.   |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted   |
| <b>Remarks</b>      | The filename (if there is one) in <i>cmdStr</i> and <i>filename</i> parameters do not have to match. |
| <b>See Also</b>     | KMGetFile  |
| <b>Example</b>      | <a href="#">Example #2</a>   |

## 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 KMErrCode
'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.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | short KMGetMaxControllerAxis(KMDevice controller);           |
| <b>Visual Basic</b> | KMGetMaxControllerAxis(ByVal controller As Long) As Integer  |
| <i>controller</i>   | Handle of the controller to be checked.                      |
| <b>Return Value</b> | Maximum number of axis allowed.                              |
| <b>Remarks</b>      | 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 KMErrCode (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];
    KMErrCode err2;
    if (IS_DEVICE_ERROR(err) // the error is device related
    {
        // This next call will eventually be KMErrGetDeviceMessage
        err2 = KMErrGetOriginalDeviceMessage(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 KMErrGetOriginalDeviceMessage"
                "[%08X].", err2);
    }
    else
    {
        // the error is an API error so get the error message
        // from the API
        err2 = KMErrGetMessage(err, strBuf, sizeof(strBuf));
        if (err2 == KM_ERR_OK)
        {
            // insert your code to print the error message here
        }
        else
            printf("Error calling KMErrGetMessage [%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 KMErrGetDeviceMessage,
        err2 = KMErrGetOriginalDeviceMessage(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 KMErrGetOriginalDeviceMessage"
        End If
    Else
        ' the error is an API error so get the error message
        ' from the API
    End If

```



```

err2 = KMErrGetMessage(err, strBuf, sizeof(strBuf))
If err2 = KM_ERR_OK
    ' insert your code to print the error message here
Else
    MsgBox "Error calling KMErrGetMessage"
End If
End If
End If

```

## KMErrGetMessage

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

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrCode KMErrGetMessage(KMErrCode errCode, LPSTR buf, short bufLen)   |
| <b>Visual Basic</b> | KMErrGetMessage(ByVal errCode As Long, ByVal buf\$, ByVal bufLen As Integer) As Long  |
| <i>errCode</i>      | Error code to get the message for.  |
| <i>buf</i>          | Character array buffer to store error message in.   |
| <i>bufLen</i>       | Length of <i>buf</i> array.   |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted  |
| <b>Remarks</b>      | 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'). |
| <b>See Also</b>     | KMErrGetDeviceMessage<br>KMErrGetOriginalDeviceMessage  |
| <b>Example</b>      | <a href="#">Example #4</a>  |

## KMErrGetDeviceMessage

Get original text and error number of last device error.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrCode KMErrGetDeviceMessage(KMErrCodePtr pErrCode, LPSTR buf, short bufLen)  |
| <b>Visual Basic</b> | KMErrGetDeviceMessage(pErrCode As Long, ByVal buf\$, ByVal bufLen As Integer) As Long  |
| <i>pErrCode</i>     | Pointer to variable to store last error number in.   |
| <i>buf</i>          | Character array buffer for device error message.   |
| <i>bufLen</i>       | Length of <i>buf</i> array   |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted   |
| <b>Remarks</b>      | Last device error is stored on a per application context basis.<br>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'). |
| <b>See Also</b>     | KMErrGetMessage<br>KMErrGetOriginalDeviceMessage   |
| <b>Example</b>      | <a href="#">Example #4</a>   |

## KMErrorGetOriginalDeviceMessage

Retrieves entire original error message from buffer.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMErrorGetOriginalDeviceMessage(LPSTR buf, short bufLen);                        |
| <b>Visual Basic</b> | KMErrorGetOriginalDeviceMessage(ByVal buf\$, ByVal bufLen As Integer) As Long                |
| <i>buf</i>          | Character array buffer to store error message in   |
| <i>bufLen</i>       | Length of <i>buf</i> array.  |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted.  |
| <b>Remarks</b>      | The text returned is the original error message that was generated and stored in the buffer. |
| <b>See Also</b>     | KMErrorGetDeviceMessage<br>KMErrorGetMessage   |
| <b>Example</b>      | <a href="#">Example #4</a>   |

### 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.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMDevice KMGetAxisController(KMDevice axis)          |
| <b>Visual Basic</b> | KMGetAxisController(ByVal axis As Long) As Long      |
| <i>axis</i>         | Handle to a axis                                     |
| <b>Return Value</b> | Handle of the controller the axis is located on.     |
| <b>Remarks</b>      | An axis is assigned to a controller during creation. |
| <b>See Also</b>     | KMCreateSercosAxis<br>KMCreateController             |
| <b>Example</b>      | <a href="#">Example #5</a>                           |

## KMGetDeviceName

Get the name assigned to the device.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMGetDeviceName(KMDevice device, LPSTR name, long nameLen)                   |
| <b>Visual Basic</b> | KMGetDeviceName(ByVal device As Long, ByVal deviceName\$, ByVal nameLen As Long) As Long |
| <i>device</i>       | Handle to a device   |
| <i>name</i>         | String buffer to place the name in   |
| <i>nameLen</i>      | Length of <i>name</i> buffer including zero terminator                                   |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted   |
| <b>Remarks</b>      | A name is assigned to a device during creation.  |
| <b>See Also</b>     | KMCreateController<br>KMCreateSercosAxis   |
| <b>Example</b>      | <a href="#">Example #5</a>   |

## KMGetDeviceProductClass

Get the product class for a device.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMProductClass KMGetDeviceProductClass(KMDevice device)  |
| <b>Visual Basic</b> | KMGetDeviceProductClass(ByVal device As Long) As Integer   |
| <i>device</i>       | Handle to the device   |
| <b>Return Value</b> | The product class of the device.   |
| <b>Remarks</b>      | See the product class description.<br>Product class is assigned by the API during device creation. |
| <b>See Also</b>     | KMCreateController<br>KMCreateSercosAxis   |
| <b>Example</b>      | <a href="#">Example #5</a>   |

## KMGetDeviceProductType

Get the product type for a device.

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMProductType KMGetDeviceProductType(KMDevice device)   |
| <b>Visual Basic</b> | KMGetDeviceProductType(ByVal device As Long) As Integer                                       |
| <i>device</i>       | Handle to the device  |
| <b>Return Value</b> | The product type of the device.   |
| <b>Remarks</b>      | See product type description.<br>Product type is assigned by the user during device creation. |
| <b>See Also</b>     | KMCreateController<br>KMCreateSercosAxis  |
| <b>Example</b>      | v   |

## KMGetDeviceCommType

Get the communication type for a device.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMCommType KMGetDeviceCommType(KMDevice device)  |
| <b>Visual Basic</b> | KMGetDeviceCommType(ByVal device As Long) As Integer   |
| <i>device</i>       | Handle to the device   |
| <b>Return Value</b> | The communication type of the device. See communication type description.                      |
| <b>Remarks</b>      | Communication type is assigned during device creation depending on what KMCreate call is used. |
| <b>See Also</b>     | KMCreateController<br>KMCreateSercosAxis   |
| <b>Example</b>      | <a href="#">Example #5</a>   |

## KMGetDeviceCommPort

Get the COM port the device is attached to.

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrorCode KMGetDeviceCommPort(KMDevice device, LPSTR commPort, long commPortLen)                        |
| <b>Visual Basic</b> | KMGetDeviceCommPort(ByVal device As Long, ByVal commPort\$, ByVal commPortLen As Long) As Long            |
| <i>device</i>       | Handle to the device  |
| <b>comPort</b>      | String with COM port that the device is connected to  |
| <b>comPortLen</b>   | Length of <i>comPort</i> string   |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted  |
| <b>Remarks</b>      | Comm port is assigned during device creation. Will return a blank (NULL) string if no COMM port assigned. |
| <b>See Also</b>     | KMCreateController  |
| <b>Example</b>      | <a href="#">Example #5</a>  |

## KMGetControllerID

Get the controller address for the controller.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | short KMGetControllerID(KMDevice controller)               |
| <b>Visual Basic</b> | KMGetControllerID(ByVal controller As Long) As Integer     |
| <i>controller</i>   | Handle to the controller                                   |
| <b>Return Value</b> | 0 is returned if there is no controller ID for the device. |
| <b>Remarks</b>      | ControllerID is assigned during controller creation.       |
| <b>See Also</b>     | KMCreateController   |
| <b>Example</b>      | <a href="#">Example #5</a>                                 |

## 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.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMWriteDeviceFile(LPSTR fileName)      |
| <b>Visual Basic</b> |  |
| <i>fileName</i>     | String with filename                               |
| <b>Return Value</b> | KM_ERR_OK if command is successfully.              |
| <b>Remarks</b>      | API must have rights to write to <i>fileName</i> . |
| <b>See Also</b>     | KMReadDeviceFile                                   |
| <b>Example</b>      | <a href="#">Example #6</a>                         |

## KMReadDeviceFile

Read a new API configuration from a file.

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrorCode KMReadDeviceFile(LPSTR fileName)        |
| <b>Visual Basic</b> |   |
| <i>fileName</i>     | String with filename                                |
| <b>Return Value</b> | KM_ERR_OK if command is successfully.               |
| <b>Remarks</b>      | API must have rights to read from <i>fileName</i> . |
| <b>See Also</b>     | KMWriteDeviceFile                                   |
| <b>Example</b>      | <a href="#">Example #6</a>                          |

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

## 2.7 Device Table Iterators

### KMCreateDeviceIterator

Create an iterator to traverse the device table.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMDeviceIterator KMCreateDeviceIterator(KMProductClass prodClass, KMDevice device) |
| <b>Visual Basic</b> | KMCreateDeviceIterator(ByVal prodClass As Integer, ByVal device As Long) As Long   |
| <i>prodClass</i>    | Product class to select an iterator for.   |



| <i>device</i>       | Device to select an iterator for.  |                                  |               |               |            |      |                                  |                  |      |                 |             |      |            |            |      |          |             |            |                             |            |            |                           |            |       |                      |
|---------------------|--|----------------------------------|---------------|---------------|------------|------|----------------------------------|------------------|------|-----------------|-------------|------|------------|------------|------|----------|-------------|------------|-----------------------------|------------|------------|---------------------------|------------|-------|----------------------|
| <b>Return Value</b> | Handle to the device iterator, NULL on failure.  |                                  |               |               |            |      |                                  |                  |      |                 |             |      |            |            |      |          |             |            |                             |            |            |                           |            |       |                      |
| <b>Remarks</b>      | <p>These combinations of <i>prodClass</i> and <i>device</i> type result in the following:</p> <table border="1"> <thead> <tr> <th><b>prodClass</b></th> <th><b>Device</b></th> <th><b>Result</b></th> </tr> </thead> <tbody> <tr> <td>CLASS_NONE</td> <td>NULL</td> <td>all controllers, groups and axes</td> </tr> <tr> <td>CLASS_CONTROLLER</td> <td>NULL</td> <td>all controllers</td> </tr> <tr> <td>CLASS_GROUP</td> <td>NULL</td> <td>all groups</td> </tr> <tr> <td>CLASS_AXIS</td> <td>NULL</td> <td>all axes</td> </tr> <tr> <td>CLASS_GROUP</td> <td>CONTROLLER</td> <td>all groups for a controller</td> </tr> <tr> <td>CLASS_AXIS</td> <td>CONTROLLER</td> <td>all axes for a controller</td> </tr> <tr> <td>CLASS_AXIS</td> <td>GROUP</td> <td>all axes for a group</td> </tr> </tbody> </table> <p>All other combinations are invalid.</p> | <b>prodClass</b>                 | <b>Device</b> | <b>Result</b> | 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 |
| <b>prodClass</b>    | <b>Device</b>  | <b>Result</b>                    |               |               |            |      |                                  |                  |      |                 |             |      |            |            |      |          |             |            |                             |            |            |                           |            |       |                      |
| 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             |               |               |            |      |                                  |                  |      |                 |             |      |            |            |      |          |             |            |                             |            |            |                           |            |       |                      |
| <b>See Also</b>     | KMDestroyDeviceIterator<br>KMGetNextDevice<br>KMGetPrevDevice<br>KMReadDeviceFile<br>KMWriteDeviceFile<br>KMCreateController<br>KMCreateSercosAxis   |                                  |               |               |            |      |                                  |                  |      |                 |             |      |            |            |      |          |             |            |                             |            |            |                           |            |       |                      |
| <b>Example</b>      | <a href="#">Example #7</a>   |                                  |               |               |            |      |                                  |                  |      |                 |             |      |            |            |      |          |             |            |                             |            |            |                           |            |       |                      |

## KMDestroyDeviceIterator

Destroy the device table iterator.

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMErrorCode KMDestroyDeviceIterator(KMDeviceIterator deviceIter)    |
| <b>Visual Basic</b> | KMDestroyDeviceIterator(ByVal deviceIter As Long) As Long           |
| <i>deviceIter</i>   | Handle for the iterator to destroy                                  |
| <b>Return Value</b> | KM_ERR_OK if command is successfully transmitted.                   |
| <b>Remarks</b>      | Must be called to release memory allocated by the API for the user. |
| <b>See Also</b>     | KMCreateDeviceIterator<br>KMGetNextDevice<br>KMGetPrevDevice        |
| <b>Example</b>      | <a href="#">Example #7</a>  |

## KMGetNextDevice

Retrieve the next device in the device table.

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | KMDevice KMGetNextDevice(KMDeviceIterator deviceIter)             |
| <b>Visual Basic</b> | KMGetNextDevice(ByVal deviceIter As Long) As Long                 |
| <b>Return Value</b> | Handle to the next device or NULL if another device is not found. |

|                 |   |
|-----------------|---|
| <b>Remarks</b>  | Must create a device iterator first with <code>KMCreateDeviceIterator</code> .                              |
| <b>See Also</b> | <code>KMCreateDeviceIterator</code><br><code>KMDestroyDeviceIterator</code><br><code>KMGetPrevDevice</code> |
| <b>Example</b>  | <a href="#">Example #7</a>  |

## KMGetPrevDevice

Retrieve the previous device in the device table.

|                     |   |
|---------------------|---|
| <b>C/C++</b>        | <code>KMDevice KMGetPrevDevice(KMDeviceIterator deviceIter)</code>  |
| <b>Visual Basic</b> | <code>KMGetPrevDevice(ByVal deviceIter As Long) As Long</code>  |
| <b>Return Value</b> | Handle to the previous device or NULL if another device is not found.                                       |
| <b>Remarks</b>      | Must create a device iterator first with <code>KMCreateDeviceIterator</code> .                              |
| <b>See Also</b>     | <code>KMCreateDeviceIterator</code><br><code>KMDestroyDeviceIterator</code><br><code>KMGetNextDevice</code> |
| <b>Example</b>      | <a href="#">Example #7</a>  |

### 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.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMAsyncHandler KMAsyncGetHandler(void)   |
| <b>Visual Basic</b> | KMAsyncGetHandler() As Long  |
| <b>Return Value</b> | Window handle (HWND) of present asynchronous message handler.                        |
| <b>Remarks</b>      | Save return value before calling KMAsyncSetHandler.                                  |
| <b>See Also</b>     | KMAsyncGetHandler<br>KMAsyncSetHandler<br>KMAsyncGetMessage<br>KMAsyncEnableMessages |
| <b>Example</b>      | <a href="#">Example #8</a>   |

## KMAsyncSetHandler

Set the asynchronous message handler.

|                      |  |
|----------------------|--|
| <b>C/C++</b>         | KMErrorCode KMAsyncSetHandler(KMAsyncHandler hAsyncHandler)  |
| <b>Visual Basic</b>  | KMAsyncSetHandler(ByVal hAsyncHandler As Long) As Long   |
| <i>hAsyncHandler</i> | Window handle (HWND) to send asynchronous messages to  |
| <b>Return Value</b>  | KM_ERR_OK if command is successfully transmitted.  |
| <b>Remarks</b>       | Call KMAsyncGetHandler and save the return value before calling KMAsyncSetHandler in order to restore prior value. |
| <b>See Also</b>      | KMAsyncGetHandler<br>KMAsyncSetHandler<br>KMAsyncGetMessage<br>KMAsyncEnableMessages                               |
| <b>Example</b>       | <a href="#">Example #8</a>   |

## KMAsyncGetMessage

Gets the message contents and message type for the asynchronous message associated with the asynchronous message handle stored in WM\_KM\_ASYNC's IPARAM.

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMAsyncGetMessage(LPARAM hAsyncMsg, LPSTR lpszMessage, UINT nMessageSize, LPSTR nMessageType)                  |
| <b>Visual Basic</b> | KMAsyncGetMessage (ByVal hAsyncMsg As Long, ByVal lpszMessage\$, ByVal nMessageSize As Long, ByVal nMessageType\$) As Long |
| <i>hAsyncMsg</i>    | Handle associated with the message (LPARAM).   |
| <i>lpszMessage</i>  | The text associated with the message.  |
| <i>nMessageSize</i> | Size of <i>lpszMessage</i> buffer.   |
| <i>nMessageType</i> | The type of message received.  |

|                     |  |
|---------------------|--|
| <b>Return Value</b> | KM_ERR_OK if message was retrieved successfully.                                     |
| <b>Remarks</b>      | This function can only be called once for each asynchronous message handled.         |
| <b>See Also</b>     | KMAsyncGetHandler<br>KMAsyncSetHandler<br>KMAsyncGetMessage<br>KMAsyncEnableMessages |
| <b>Example</b>      | <a href="#">Example #8</a>   |

## KMAsyncEnableMessages

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

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMAsyncEnableMessages(UINT16 bVal)                                       |
| <b>Visual Basic</b> | KMAsyncEnableMessages (ByVal bVal As Integer) As Long                                |
| <i>bVal</i>         | Non zero means display the asynchronous messages.                                    |
| <b>Return Value</b> | KM_ERR_OK if function was executed successfully.                                     |
| <b>Remarks</b>      |  |
| <b>See Also</b>     | KMAsyncGetHandler<br>KMAsyncSetHandler<br>KMAsyncGetMessage<br>KMAsyncEnableMessages |
| <b>Example</b>      | <a href="#">Example #8</a>   |

## 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.

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

|                 |   |
|-----------------|---|
| <b>See Also</b> | KMTCPGetNumDevices<br>KMTCPGetDeviceInformation |
| <b>Example</b>  | <a href="#">Example #9</a>                      |

## KMTCPGetNumDevices

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

|                     |  |
|---------------------|--|
| <b>C/C++</b>        | KMErrorCode KMTCPGetNumDevices(LPUINT16<br>lpNumDevices)                                     |
| <b>Visual Basic</b> | KMTCPGetNumDevices(lpNumDevices As Integer) As Long  |
| <i>lpNumDevices</i> | Pointer to a 16 bit integer to receive the number of devices                                 |
| <b>Return Value</b> | KM_ERR_OK on success   |
| <b>Remarks</b>      | The count returned by KMTCPGetNumDevices is updated each time KMTCPRefreshDevices is called. |
| <b>See Also</b>     | KMTCPRefreshDevices<br>KMTCPGetDeviceInformation   |
| <b>Example</b>      | <a href="#">Example #9</a>   |

## KMTCPGetDeviceInformation

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

|                        |  |
|------------------------|--|
| <b>C/C++</b>           | KMErrorCode KMTCPGetDeviceInformation(LPSTR<br>lpzNameArray[], LPSTR lpzAddressArray[], LPSTR<br>lpzSNArray[], UINT16 DIPArray[], INT32 nNumDevices)   |
| <b>Visual Basic</b>    | KMTCPGetDeviceInformation(ByVal lpzNameArray As<br>Long, ByVal lpzAddressArray As Long, ByVal lpzSNArray<br>As Long, ByVal nNumDevices As Long) As Long  |
| <i>lpzNameArray</i>    | An array of pointers to strings containing the names of the controllers.   |
| <i>lpzAddressArray</i> | An array of pointers to strings (at least 16 characters long) containing the IP addresses of the controllers.  |
| <i>lpzSNArray</i>      | An array of pointers to strings containing the serial numbers of the controllers.  |
| <i>DIPArray</i>        | An array of short integers (16-bit) containing the DIP switch addresses of the controllers.  |
| <i>nNumDevices</i>     | The length (number of elements) of <i>lpzNameArray</i> , <i>lpzAddressArray</i> and <i>lpzSNArray</i> .  |
| <b>Return Value</b>    | KM_ERR_OK on success   |
| <b>Remarks</b>         | 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.<br><br>Visual Basic is somewhat unusual in that the function passes arrays of strings. <a href="#">Example #9</a> shows how to use this function. |

|                 |   |
|-----------------|---|
| <b>See Also</b> | KMTCPGetNumDevices<br>KMTCPRefreshDevices |
| <b>Example</b>  | <a href="#">Example #9</a>                |

## 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

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