

softMC Training – Module 10

# Servotronics Motion API



# Contents

- Initialization
- Device Table
- Accessing Devices
  - Command Execution
  - Variable Access
- Error Messages
  - Error Codes
- Termination (Cleanup)
- Sending/Retrieving Files
- Advanced Asynchronous Message Handling
- Programming with Visual Basic

# OVERVIEW

# Application Programming Interface (API)

- The **Servotronix Motion API** provides the means for interfacing your machine controller with the **softMC** motion controller :
  - 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.
- The API is provided as a Windows DLL, named **KMAPI.dll**, making it accessible from most Windows programming languages.
- The API is installed as part of the **ControlStudio** installation.
- The API is connected to an Entry Station instance in the softMC. The system has a limit of 3 entry station instances; that is, no more than 3 connections from 3 different computers.

# INITIALIZATION

# Initialization

- To start using the API, call the initialization function **KMInitialize**.
- Initialization ensures that the data maintained internally by the API about your application is correct.

# DEVICE TABLE

# Device Table

- A device table is a database managed by the API.
- Table is populated with devices by the programmer
- Allows programmer to execute commands on any device in the device table
  
- **KMWriteDeviceFile** and **KMReadDeviceFile** functions are used to save and load device table

# Adding Devices to Device Table

- After initialization, the next task is to describe and add to the API device table the devices you want to control.
  - Typical system: a softMC motion controller with several connected CDHD servo drives.
- **KMCreate** ... functions allow you to add devices to the API device table.
  - KMCreate functions return a **handle** to a newly created device.
  - A handle allows the API to know which device you are referring to.
- **KMGet** ... functions retrieve a handle or information about a particular device.
- **KMDestroyDevice** function removes devices from the device table.
- **KMCreateSercosAxis** returns a handle to an **axis**.

# Device Table Iteration

- **KMCreateDeviceIterator** function enables iteration through the device table
- An iterator can display all the controllers, axes, or groups, or any combination of the three.
- **KMGetNextDevice** and **KMGetPrevDevice** functions are used to move to the next or previous device on the iterator list.

# ACCESSING DEVICES

# Accessing Devices

- The API has two mechanisms for communicating with devices:
  - Command execution
  - Variable access

# Command Execution

- Command execution is handled through two functions:
  - **KMExecuteCmd**
  - **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** uses a buffer to store the response from the softMC; when responding to the command, it also indicates the length of data that has been put into the buffer.

# Variable Access

- 2 types of variable access functions
  - Functions to get and set variables for each type (long, double, string)
  - Functions to access the two components of an axis (drive and controller)
- Access string type variables on the softMC via the function **KMVariableControllerGetStringValue**

# ERROR (Status) MESSAGES

# Error Messages

- Most API functions return a(n error) code that indicates the status of the action requested by the programmer.
- Successful API functions return `KM_ERR_OK`.
- Textual descriptions are often associated with the error and can be accessed via API calls.

# Error Messages

- Errors originate in the device itself or the API.
- When the message originates within the **API**, an error number is returned and can be compared against the list of errors in the API header files (C/C++) or the global files (Visual Basic).
- 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**.
- The text of the error message sent by the device can be retrieved via **KMErrorGetOriginalDeviceMessage**.

# TERMINATION

# 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.

# SENDING/RETRIEVING FILES

# Sending/Retrieving Files

- For sending and retrieving files, use functions **KMPutFile** and **KMGetFile**.
- When calling either of these functions, the path of the file and the actual command must be sent, in addition to the device handle parameter.

# ADVANCED ASYNCHRONOUS MESSAGE HANDLING

# Asynchronous Messages

- Controller-generated messages not related to a specific command are called asynchronous messages.
- Examples of asynchronous messages:
  - Over-speed warnings
  - Servo drive faults such as limit switch closures
  - Runtime error messages from softMC tasks
- When these messages are received, the API converts them into Windows messages (events) that can be programmed for delivery to the application.

# Asynchronous Messages

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

# Asynchronous Messages

- Like all Windows messages, WM\_KM\_ASYNC has two parameters:
  - **wParam**
  - **lParam**
- lParam contains a handle to the buffer that contains the asynchronous message received by the API.
- 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**.

# VISUAL BASIC PROGRAMMING

# Visual Basic Programming

- The API is a set of DLLs.
- When DLLs return strings to Visual Basic they may appear to be corrupted, especially if there was data in the string before it was passed to the DLL function.
  - Caused by differences in the method of storing strings used VB and the method the DLLs use (“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, scan the string for CHR\$(0), and then trim the characters to the right.
- Alternately, make sure the string is clear before passing it to a DLL, by assigning it to vbNullString prior to calling the DLL function.

**END**