

## ***PORTING APPLICATIONS***

### **Porting 16 Bit Applications to 32 Bit Applications**

Over the past few years, more and more users have upgraded their systems to keep pace with the improvements in PC technology and the modern operating systems. In addition, some users have to change Motion Controllers because their product has become mature, is obsolete or no longer can be manufactured. At this point the migration issues become critically important. This addresses these issues, so that the users can make the decisions and desired results are achieved.

This document discusses some of the key issues that must be addressed when porting 16 bit applications that control motors, via an Oregon Micro Systems motion controller, to 32 bit applications.

In the past, these 16 Bit Applications were generated, using DOS or Windows 3.1. These Applications created for Windows 3.1 normally use OMS software such as the OMSWLIB and OMSWLIBM DLLs.

The following items should be considered:

- These applications will run under Windows 95 and 98, but they will NOT run under Windows NT or Windows 2000. This is because Windows 95/98 provide support for the 16 bit environment, while Win 2000 and WinNT do not.
- The applications must be maintained using language development systems that support 16 bit code generation. This makes them incompatible with the newer Oregon Micro Systems 32 bit DLLs and OMS Device drivers.
- It should also be noted that there are differences between the "services" provided by the old 16 bit DLL environment, and the current 32 bit device drivers and corresponding support DLL combinations. ———:
- ❖ **In the 16 bit environment, a "DLL" contains both the DLL code, and the code that services the controller interrupts.** (How this works is explained below).
  - Access to the controller is initiated via a call to the "OMSInitialize" function. In order to work properly, this function must have the following information provided to it :
    1. The controller's base address.
    2. An IRQ number.
    3. A "Handle" to the Window that will be receiving the call back messages that are generated by the interrupt service routine (ISR). Initial DONE\_BIT and ERROR\_BIT handling must occur in this window. (A 16 bit Visual Basic control is available to handle the call back messages.) The following controller events will generate a call back message:
      - An axis done interrupt.
      - A command error.
      - An axis over travel.
      - An encoder slip error.
    4. A unique windows message code for use by the interrupt service routine.
  - The "OMSSendString" function is used to send command strings to the controller.
  - The "OMSSendAndGetString" function is used to send query commands to the controller.
  - The "OMSClose" function is used to disconnect the DLL's interrupt service routine from the IRQ line, and to release any memory allocated by the "OMSInitialize" function.

## ❖ The 32 bit environment is a combination of a 32 bit “protected mode device driver” and a 32 bit driver support DLL.

- The driver is a “32 bit kernel mode” driver. It differs from the 16 bit environment as follows:
  - A. It must be installed prior to use.
  - B. Controller status information is captured by the driver’s interrupt service routine, and is held until the application requests it. Note this differs from the 16 bit environment where the application is notified of status changes via a windows message.

### The 32 bit driver support DLL provides the following services:

- Access to the controller via a file handle provided by the “GetOmsHandle” function.
- The “CloseOmsHandle” function releases the file handle. Note: it differs from the 16 bit “OMSClose” function in that it does not necessarily release resources or disconnect the controller from the IRQ line.
- The “SendString” function is used to send non query command strings to the controller.
- The “SendAndGetString” function is use to send query command string to the controller and to return the response string to the application.
- The “GetDoneFlags” function is used to acquire the status of the AXIS\_DONE flags that have been captured by the driver’s interrupt service routine. Note that once the DONE flags are set, they are latched by the driver until they are cleared by the application. (See the “ClrDoneFlags” function described below. Note: this method of obtaining done flag information differs from the 16 bit environment in that **the 32 bit application must poll the driver to obtain axis done flag information.**
- The “ClrDoneFlags” function is used to clear selected axis done flags.
- The “GetStatusFlags” function is use to acquire controller status flag information that has been captured by the driver’s interrupt service routine. Note: once set, a status flag is latched by the driver until it is cleared by the application. See the “ClrStatusFlags” function described below. Note: this method of obtaining status information differs from the 16 bit environment in that the 32 bit application must poll the driver to obtain status information. The following status information is latched by the 32 bit driver:
  - A command error.
  - An axis over travel.
  - An encoder slip error.
- The “ClrStatusFlags” function is used to clear selected status flags.

### APPLICATIONS THAT WERE CREATED FOR WINDOWS 3.1 OR DOS THAT USE POLLING TECHNIQUES:

- These applications will run under Windows 95 and 98 but they will not run under Windows NT or Windows 2000.
- The applications must be maintained using language development systems that support 16 bit code generation. This makes them incompatible with the newer Oregon Micro Systems 32 bit DLLs and device drivers.
- The 32 bit device drivers do not support polling techniques. 16 bit polled mode applications that are being ported to a 32 bit environment must be rewritten to support interrupt based techniques.