From 72103.2235@compuserve.com Sat Sep 10 17:30:53 1994
Received: from arl-img-1.compuserve.com by flash.cmotor.com with SMTP id AA06838
(5.65c/IDA-1.5 for <stuartg@cmotor.com>); Sat, 10 Sep 1994 17:30:49 -0700
Received: from localhost by arl-img-1.compuserve.com (8.6.4/5.940406sam)
id UAA20597; Sat, 10 Sep 1994 20:38:03 -0400
Date: 10 Sep 94 20:34:49 EDT
From: Dave Brown <72103.2235@compuserve.com>
To: Marc McClung <76336.3110@compuserve.com>
Cc: Jay Clark <72234.627@compuserve.com>, Stuart Goodnick <stuartg@cmotor.com>
Subject: WOSA/XMC Function Evolution
Message-Id: <940911003449_72103.2235_GHB81-1@CompuServe.COM>
Status: R

Hi Marc,

If you have any specific suggestions or questions on any of the functions or OLE 2.0 interfaces within the WOSA/XMC software specification please let me know. Also, if you feel there are any confusing areas or technologies described in the specification, let me know and I will try to either clarify or add more detail to the current document in those locations.

For a general time line, I would like to complete the third revision to the specification by October 1.

In the mean time, I will be working on the lower level, hardware dependent software.

Feel free to either call or email me any time. I check my email daily.

Best Regards,

Dave Brown
ROY-G-BIV Corporation

From 72103.2235@compuserve.com Tue Sep 20 13:56:44 1994
Received: from dub-img-1.compuserve.com by flash.cmotor.com with SMTP id AA01412
(5.65c/IDA-1.5 for <stuartg@cmotor.com>); Tue, 20 Sep 1994 13:56:29 -0700
Received: from localhost by dub-img-1.compuserve.com (8.6.4/5.940406sam)
id RAA22346; Tue, 20 Sep 1994 17:03:48 -0400
Date: 20 Sep 94 16:57:08 EDT
From: Dave Brown <72103.2235@compuserve.com>
To: Marc McClung <76336.3110@compuserve.com>
Cc: Jay Clark <72234.627@compuserve.com>, Stuart Goodnick <stuartg@cmotor.com>
Subject: RE: WOSA/XMC, Draft 2, Section 3.2
Message-Id: <940920205707_72103.2235_GHB35-2@CompuServe.COM>
Status: R

Hi Marc,

I am glad to see that you have had a chance to review the specification. With regard to your response, I will try to answer your questions as best as possible.

>>The good news is that Bob and Kevin have given me permission to spend the
>>next couple of weeks thoroughly reviewing the WOSA/XMC spec and providing
>>the feedback that you need.

Excellent! If possible I would like to prepare another revision of the specification and send it out between October 1 and October 15. In the mean time, your input will be very helpful.

```
>>Step 1.  In this step I understand that the Driver Administrator is an
>>         independent Window's application that the user runs to add,
>>         configure, and remove Motion Control Drivers.  I am assuming that
>>         it is very similar to the printer driver dialog that you can
>>         launch from the Windows 3.1 Control Panel program group. The
>>         printer driver dialog allows the user to select only one printer
>>         as the default printer.  Will the Driver Administrator allow the
>>         user to select more than one motion control device?  We have a
>>         few customers who use several of our products in one application.
```

Yes, we can build multi driver support into the Driver Administrator, though such support may not make it into the initial version of the product for it may incur a lot of development time. Even so, I will start designing the system for supporting multi drivers running simultaneously.
NOTE: The Revision #2 of the WOSA/XMC document does NOT support simultaneous control of multiple drivers.

In your view, how essential is the feature of simultaneously supporting multi drivers?

```
>>         Also, does the Driver Administrator application create a Driver
>>         Administrator object at this time, or is that actually done in
>>         Step 5?
```

Yes, the Driver Admin object is actually implemented within the Driver Administrator application. So, when the Driver Administrator application is invoked by the user, the Driver Admin object becomes live and is ready for the Moton Control Component to use. On the other hand, the Motion Control Component can actually load the Driver Administrator (if not loaded) and query it for the current driver(s) to use. When this occurs, the Driver Administrator runs as a hidden application.

```
>>Step 2.  What exactly is the Windows Registry?  WIN.INI? SYSTEM.INI?
>>         REG.DAT?
```

The Windows Registry is a new technology introduced in Windows NT and Chicago. The registry is also backwards compatible with Windows 3.1 via REG.DAT. Although, the Windows 3.1 registry is more limited than the implementation in NT or Chicago. In Windows NT and Chicago, the registry acts as (and replaces) all .INI files. Special hardware setup information and OLE information are some examples of other information included within the registry. In a sense, the registry is a database describing the complete system including hardware and software.

```
>>Step 5.  In this step the Motion Control Component queries (via
>>         QueryInterface?) the Driver Administrator for a pointer (pointer
>>         to which interface?) to the currently selected Motion Driver.
>>        In other words, the Motion Control Component tells the Driver
>>         Administrator to create a Motion Driver object and to pass a
>>         Motion Driver interface pointer back to the Motion Control
>>         Component?
```

QueryInterface is an OLE function supported by every object. Actually, every OLE object supports a base interface called IUnknown. IUnknown implements the following three methods:

```
        QueryInterface  - used to get a pointer to other interfaces.
        AddRef          - used to increment the objects reference.
        Release         - used to decrement the objects reference and free the
object
```

if its reference is 0.

Once a pointer to the IUnknown interface is aquired, the holder of the pointer can aquire a pointer to any of the other interfaces exposed by the object. Since this interface is general, the Driver Admin object doesn't need to know about any of the Driver's specific interfaces. All that the Driver Admin cares about is creating the driver corresponding with a particular device driver DLL. >From that point on, the Driver Admin only knows about an IUnknown pointer associated with the DLL implementing the real Driver object. That pointer to IUnknown is the same one returned to the Motion Control Component when it queries the Driver Admin for the current driver to use.

>>Step 6. What does the Motion Control Component do with the pointer to
>> the Motion Driver's IUnknown interface? Or is this just a
>> byproduct of object creation?

Once the Motion Control Component has the pointer to the Motion Driver's IUnknown interface, it queries each interface, exposed by the driver, as needed. For example, when the application, using the component, directs the component to perform an absolute move (See Section 3.5), the Motion Control Component first queryies the Motion Control Driver for its IDrvCore_Motion interface (See Section 9.4.7). Depending on whether the driver is run in Servo or Stepper mode the IDrvCore_Servo or IDrvCore_Stepper interface will be queried for by the Motion Control Component. After taking care of all state checking and parameter validation, the Motion Control Component then performs the move using the Driver's motion interface pointer (Servo or Stepper) by calling its MoveAbs() method (See Section 9.4.7).

Note: The reason for having both a IDrvCore_Servo and IDrvCore_Stepper interface is to support hardware that one driver will be able to run hardware that may be run in servo or stepper mode.

Note: The Tune method will be added to the IDrvCore_Stepper interface. Even though the function will only be a stub that returns an error for it will not be implemented, its existance will allow the IMotion interface to treat the stepper and servo interfaces in the exact same manner. The only difference between the two will be their implementations and their interface ID's. Having two different interface ID's will allow one object to support both implementations.

>>Step 7. Is the ICurrentState interface described here the same as the
>> IUnknown interface described in Step 6?

No. Like all other interfaces in OLE, the ICurrentState interface "inherits" from the IUnknown interface. The ICurrentState interface is used to query a snap-shot of all state information regarding the specific motion control subsystem managed by the Motion Control Driver. The ICurrentState and IDrvCore_CurrentState operate independently from all other interfaces, except when . For example, when using the IDrvExt_Motion interface (See Section 9.5.9) to get the maximum speed, by calling GetMaxSpeed(), the IDrvExt_Motion interface would go directly to the hardware and query the value. Later on in the controlling application, the user may want to view or set, the hardware state. To view the complete state they would use the ICurrentState interface.

>>In Section 4.0 you show how a C application can create the Motion Control
>>Component and how to get at IMotion and IEncoder. Its still not clear
>>to me when the Motion Driver object gets created? Does the Motion Control
>>Component automatically create it via the Drive Administrator?

Refering back to section 3.2 Initialization (Core MCAPI), in step 5, the Motion Control Component queryies the Driver Admin for the current driver to use. At this point, the Dirver Admin creates the Driver object corresponding to the

current driver selected by the user (or the default driver). Once created, the driver object's IUnknown pointer is passed back to the Motion Control Component.

>>Shouldn't the C programmer should have direct control over creation of the Motion
>>Driver object? I could imagine wanting to create a Motion Driver object
>>for a servo card at I/O address 768 and interrupt IRQ10. In the same
>>application another Motion Driver might be created for a stepper card
>>at I/O address 800 and interrupt IRQ11.

I understand. All driver specific settings, such as the I/O address and IRQ used by the motion control hardware (and Driver) are set using the IDrvExtUI_ICurrentState::Initialize() method (See Section 9.6.2). Either the Driver Admin or the Motion Control Component can invoke this method. The Driver Admin invokes this method when the user selects a "Setup..." button for the current driver. And, the Motion Control Component invokes this method when the ICurrentState::InitializeEx() method (See Section 8.3.3) is called. Note, the IDrvExtUI_ICurrentState::Initialize() method will display a driver specific dialog.

You bring up a good point that the C++ developer should be able to specify the I/O port and IRQ used. Once we do this, though, the hardware independence shield will start to break down, for each application using the hardware will now have direct access to the IRQ's and IO ports used by the motion control hardware. Instead of configuring each driver from within each application that uses the driver via the Motion Control Component, all driver configurations, such as I/O ports and IRQs used should be setup via the Driver Admin or Motion Control Component's InitializeEx() method. The same is true when programming a printer or when programming to ODBC. When programming a printer, the application is not required to tell the print sub-system whether or not the printer is using a Serial connection or Parallel connection, instead the application asks the print subsystem to give it a connection to a printer. For example, the application would tell the print sub-system to give it a connection to "Printer A". Once the connection is made to "Printer A", the application can direct the printer to set itself up. If the application attempts to print without setting up the printer the printer object will error letting the application know that they need to notify the user to configure the printer before printing.

We want to do the same with motion control hardware where the application queryies the Driver Admin for a connection to the "Motion Controller A", which is made through a pointer to the "Motion Controller A" Device Driver objects IUnknown interface. If the application attempts a move or some other motion control action, and the user has not specified the IRQ or IO port yet, the driver will return an error to the component, who will return the error to the application. Once receiving the error, the application should notify the user that the motion control system needs to be configured. At this time, the user should set the IRQ and IO port info for the driver either through the Driver Admin or through the application. To have the application configure the driver, it will need to supply some user interface object, such as a button or menu item, that then triggers code that calls the IMotion::InitializeEx() method which then calls the IDrvExtUI_CurrentState::Initialize() method which actually pops up the Motion Control Driver's configuration dialog box.

Note, if a driver specific IDrvExtUI_CurrentState::Initialize() method is not supported, the Driver Stub IDrvExtUI_CurrentState::Initialize() method will be called to display a default configuration dialog box. The default configuration dialog box will probably only contain fields allowing the user to set an IRQ and IO port.

I hope this information helps. The points you are bringing up will be very helpful for the next revision of the specification. If you have any other

questions on specification or on this email, please let me know.

Dave Brown
ROY-G-BIV Corporation

From 72103.2235@compuserve.com Tue Sep 20 13:56:45 1994
Received: from dub-img-2.compuserve.com by flash.cmotor.com with SMTP id AA01414
(5.65c/IDA-1.5 for <stuartg@cmotor.com>); Tue, 20 Sep 1994 13:56:31 -0700
Received: from localhost by dub-img-2.compuserve.com (8.6.4/5.940406sam)
id RAA08245; Tue, 20 Sep 1994 17:03:49 -0400
Date: 20 Sep 94 16:57:15 EDT
From: Dave Brown <72103.2235@compuserve.com>
To: Marc McClung <76336.3110@compuserve.com>
Cc: Jay Clark <72234.627@compuserve.com>, Stuart Goodnick <stuartg@cmotor.com>
Subject: WOSA/XMC, Draft 2, Section 3.3
Message-Id: <940920205714_72103.2235_GHB35-3@CompuServe.COM>
Status: R

Hi Marc,

See below for response to your email:

>>For this correspondence I would like to brainstorm Section 3.3 that
>>describes a scenario for code generation as it relates to
>>"Initialization Tuning". However, I am not familiar with with the
>>concept of "Initialization Tuning" in our products. Also, I wasn't
>>sure which MCAPI code generation interface this refers to, although
>>I did see an Initialize method in the ICurrentState interface.

In the scenario map discussed in section 3.3 the application contains the initialization tuning algorithm and the Motion Control Component is used to generate the code for the tuned initialization process. I came up with this example after reviewing the Motion Architecture product. I wasn't sure whether Motion Architecture tuned the initialization hardware dependent code or just helped the user generate it. Anyways, in section 3.3 the application could be one just like Motion Architect, but instead of directly generating the Compumotor specific motion control codes, the application would use the Motion Control Component interfaces and their methods to generate the initialization code. Doing so would make the Motion Architect hardware independent.

>>While we're on the subject of code generation interfaces, I think
>>there is a better way of implementing code generation than using
>>code generation interfaces. How about giving each interface the
>>capability of doing code generation. For example, the C code from
>>Section 4 (pMove->Interpolated(3, ad)) could either perform the
>>interpolated move or generate code for the interpolated move and save
>>it to a file. At the MCAPI level code generation could be turned on/off
>>via a flag passed to an interface method.

That is the same way I was thinking code generation would work. Section 8.4 talks about this briefly. The general Motion Control Component interfaces, such as ITimer, IMotion, etc, may be run in one of the three modes: real-time, code generation, or mixed. When run in code generation mode, code will be generated and streamed out to either the personality section within the registry, to file, or to some other user specified location. The code generation specific interfaces are provided to allow applications to write complete motion control programs. In a way these interfaces allow the application developer to write an interpreter or compiler for the motion control hardware, but in a hardware independent manner. This area of the component will be under investigation for a little while for I am not sure that all hardware implementations will support all of the code generation methods provided. If hardware does not support a