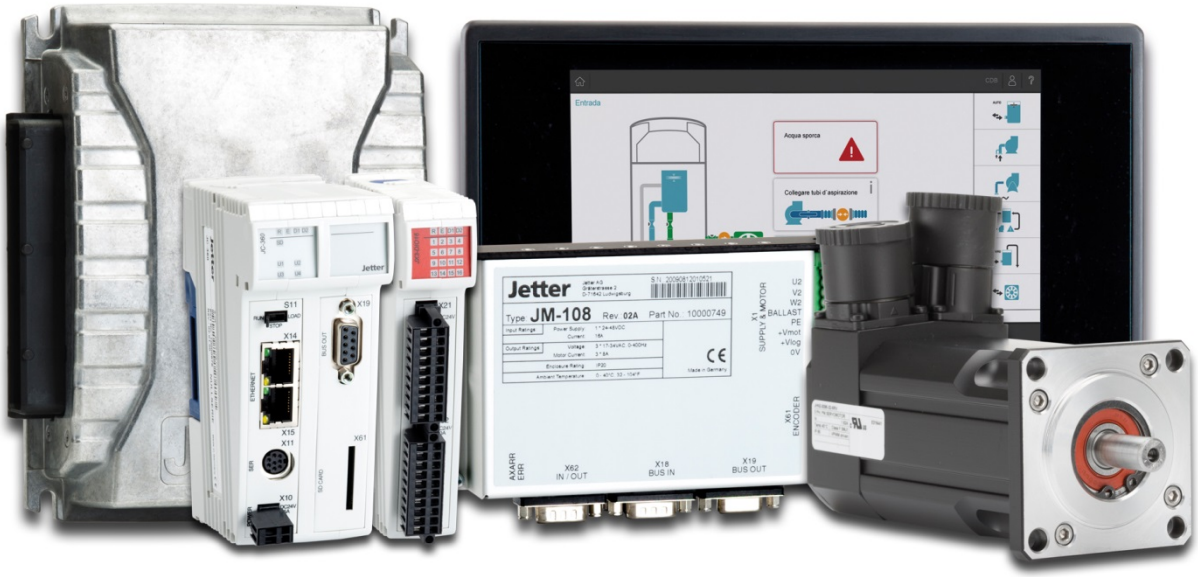
Jetter automation



MCX - Diagnostics

Application Note 055

608 860 68_00

We automate your success.

This document has been compiled by Jetter AG with due diligence based on the state of the art as known to them. Any revisions and technical advancements of our products are not automatically made available in a revised document. Jetter AG shall not be liable for any errors either in form or content, or for any missing updates, as well as for any damage or detriment resulting from such failure.

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg
Germany

www.jetter.de

Phone:

Switchboard	+49 7141 2550-0
Sales	+49 7141 2550-531
Technical hotline	+49 7141 2550-444

E-mail:	info@jetter.de
Technical hotline	hotline@jetter.de
Sales	vertrieb@jetter.de

Product name	MCX - Diagnostics
Document type	Application Note 055

Translation of the original German language document

Document revision	1.00
Item number	60886068_00
Date of issue	2021-05-21

Table of Contents

1 Introduction

Programming, commissioning, testing and operation of an application by means of MCX also involves the use of the diagnostic tools provided by the MCX and the JetSym programming environment.
This includes not only the evaluation of errors, but also the testing of the desired behavior or program flow.
The purpose of this application note is to show which diagnostic tools are offered by JetSym and how they are used. Furthermore, the various means of analysis within an application program are presented.

2 Types of Errors

2.1 User Errors

Incorrect operation of the MCX, e.g. specification of invalid values, travel commands for inactive objects, ..

2.2 Axis Errors

Errors that have occurred on the drive, e.g. following error, DC link overvoltage, encoder error, …

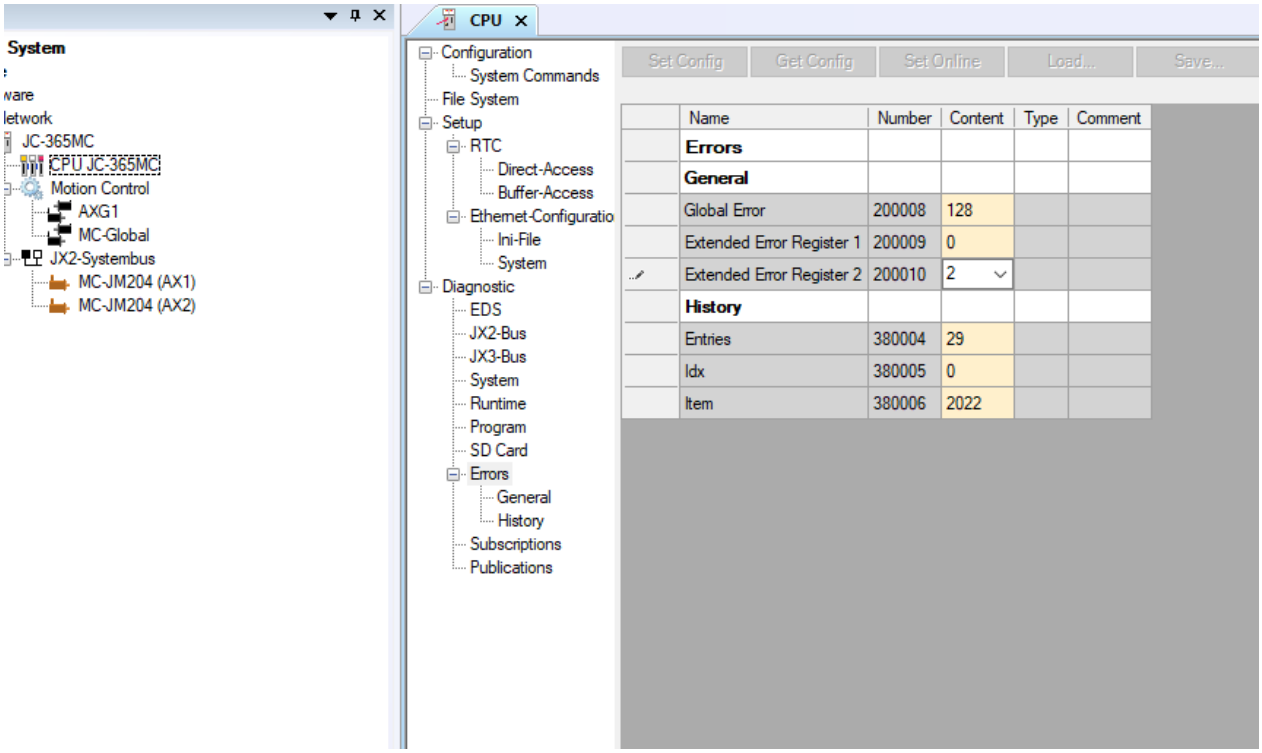
2.3 Group Errors

If at least one axis error occurs in an active group, this becomes a group error.

3 JetSym

3.1 Controller - Checking for system errors

Specific system registers can be displayed in the hardware setup window of the set controller.

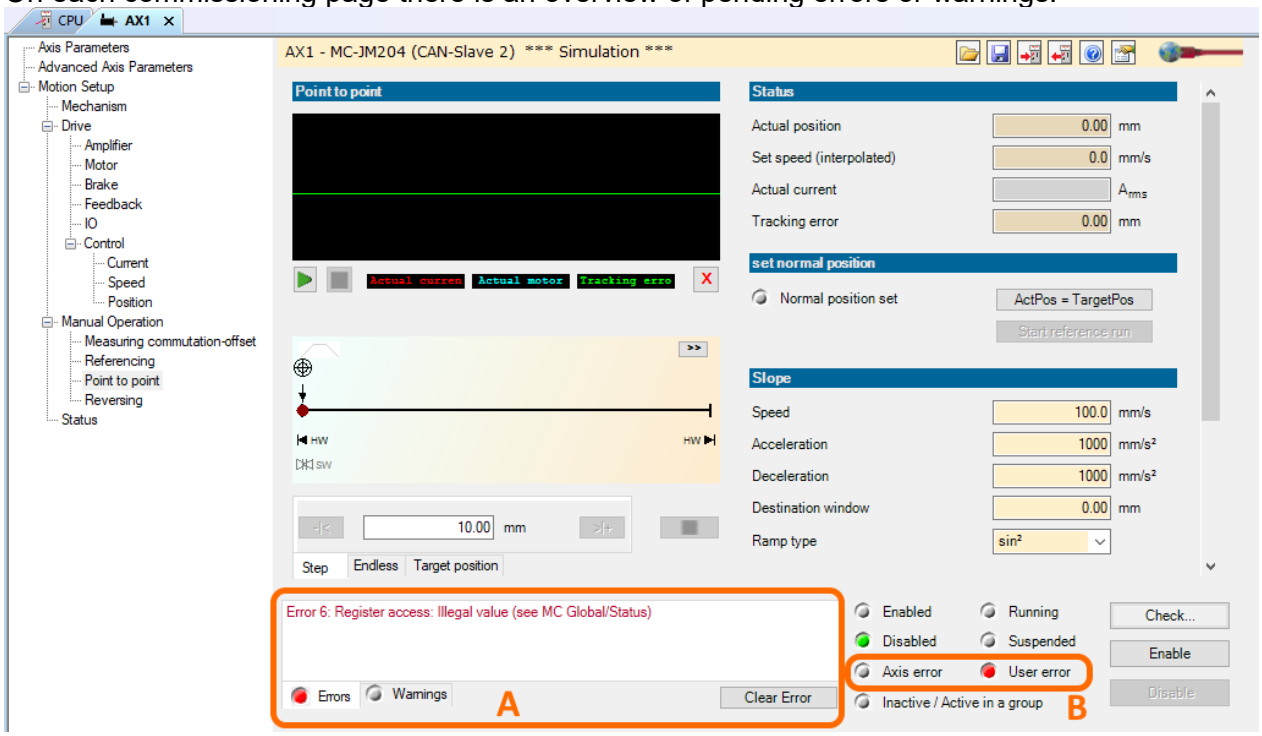


These also include detected system errors:
Register 200008: Error register
Bit 7: Error in extended error register
Register 200010: Extended error register 2
Bit 1: Error in the MCX object

A detailed description of the additional bits in these registers can be found in the manuals of the respective controller.
Bit 7 in register 200008 only indicates that errors are displayed in the extended error registers, not that an error has been detected in the MCX.
Bit 1 in the register is set as soon as the MCX has detected an error. Clearing the error with Clear-Errors() does not reset this bit. It must be reset specifically. The associated bit 7 in register 200008 must also be reset separately. If other errors are displayed in the extended error registers, bit 7 in register 200008 remains set.

3.2 Motion Setup - Single Axis

On each commissioning page there is an overview of pending errors or warnings.

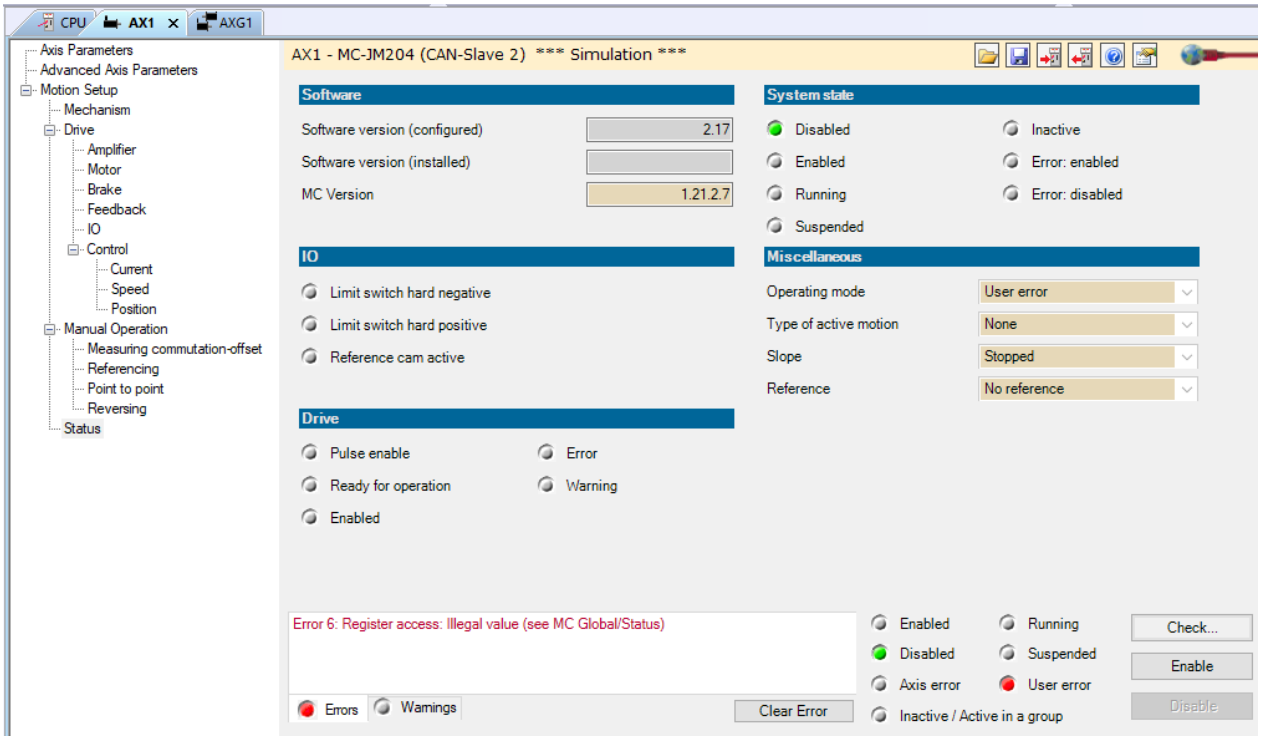


If errors or warnings are present, they are signaled by the red display of the respective tab.
If the "Errors" tab is active, errors affecting the given axis are listed in area A. Likewise, when the "Warnings" tab is selected, the pending warnings are displayed.
In addition, in area B is displayed whether an axis or user error is active.
The "Clear Error" button can be used to acknowledge pending errors.

3.2.1 Status Overview - Single Axis

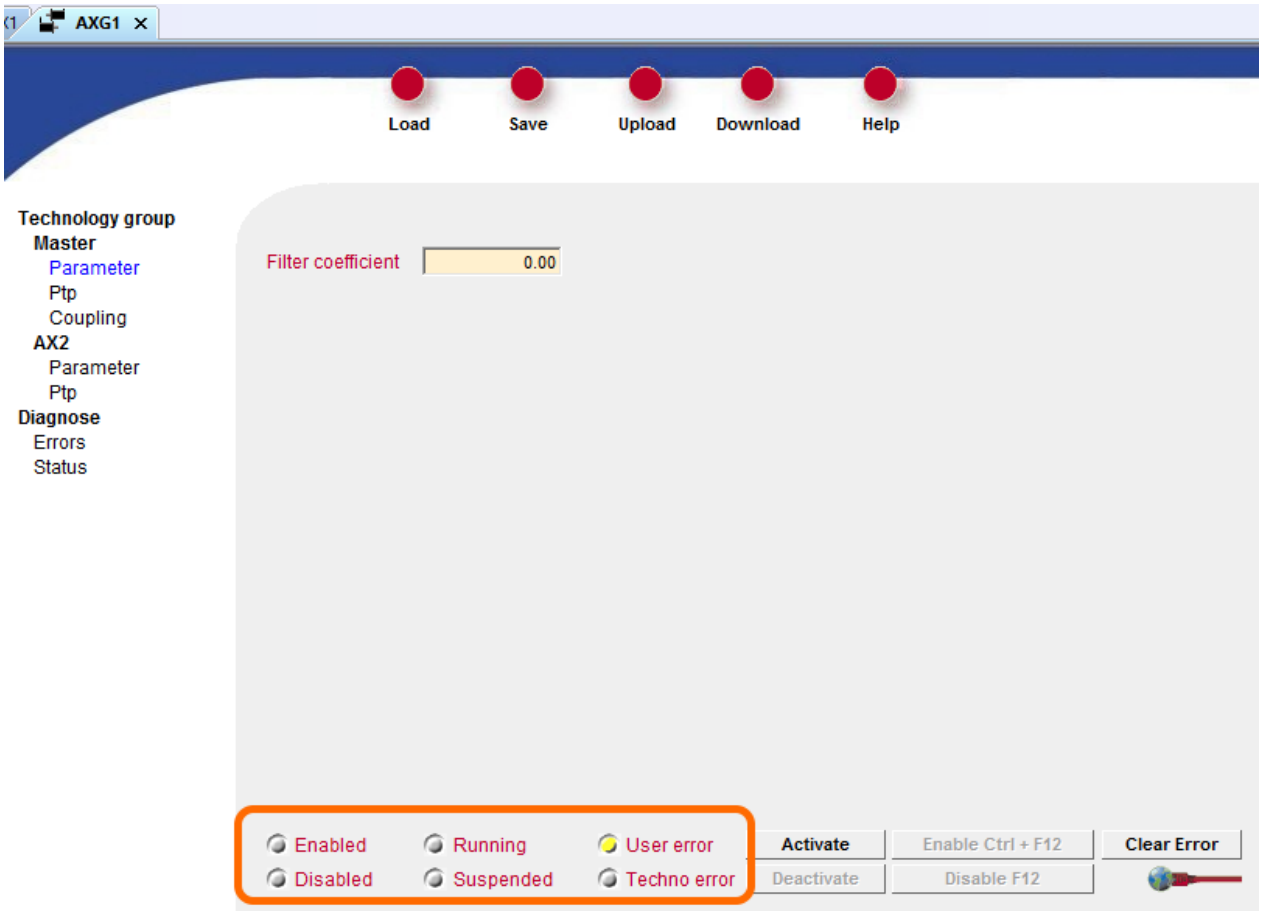
This overview page displays important status information about a single axis.

- Operating system version of the drive
- MCX version
- Switches
- Hardware status of the drive
- Display of MCX states
- Miscellaneous information
    - Operating mode
    - Type of active motion
    - Ramp (slope) status
    - Reference status


CPU
AX1
AXG1
Axis Parameters
Advanced Axis Parameters
Motion Setup
Mechanism
Drive
Amplifier
Motor
Brake
Feedback
IO
Control
Current
Speed
Position
Manual Operation
Measuring commutation-offset
Referencing
Point to point
Reversing
Status
AX1 - MC-JM204 (CAN-Slave 2) *** Simulation ***
Software
Software version (configured)
2.17
Software version (installed)
MC Version
1.21.2.7
System state
Disabled
Inactive
Enabled
Error: enabled
Running
Error: disabled
Suspended
IO
Limit switch hard negative
Limit switch hard positive
Reference cam active
Miscellaneous
Operating mode
User error
Type of active motion
None
Slope
Stopped
Reference
No reference
Drive
Pulse enable
Error
Ready for operation
Warning
Enabled
Error 6: Register access: Illegal value (see MC Global/Status)
Enabled
Running
Disabled
Suspended
Axis error
User error
Inactive / Active in a group
Check...
Enable
Disable
Errors
Warnings
Clear Error

3.3 Motion Setup - Group

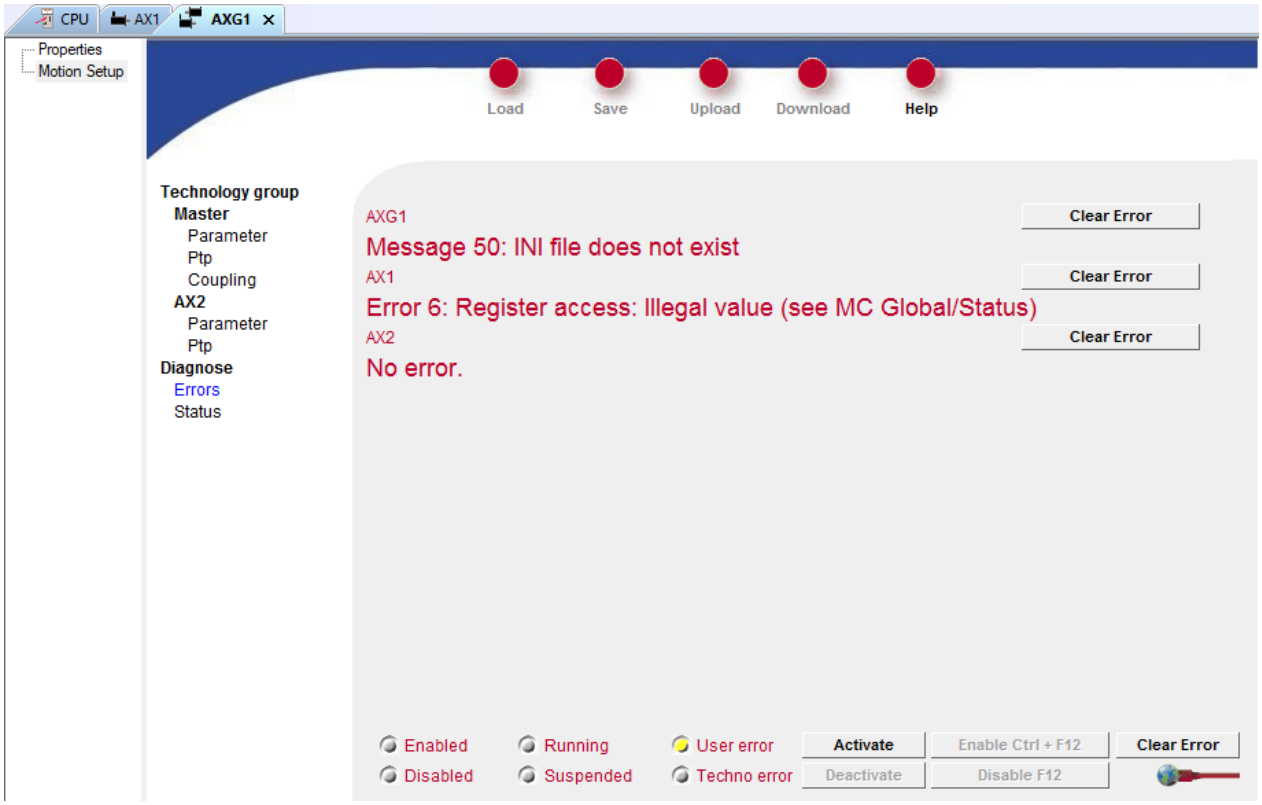
On the commissioning page of an axis group, the active status of the group is displayed as well as whether there is a user error or a group error.



The "Clear Error" button can be used to acknowledge pending errors.

3.3.1 Error Overview - Group

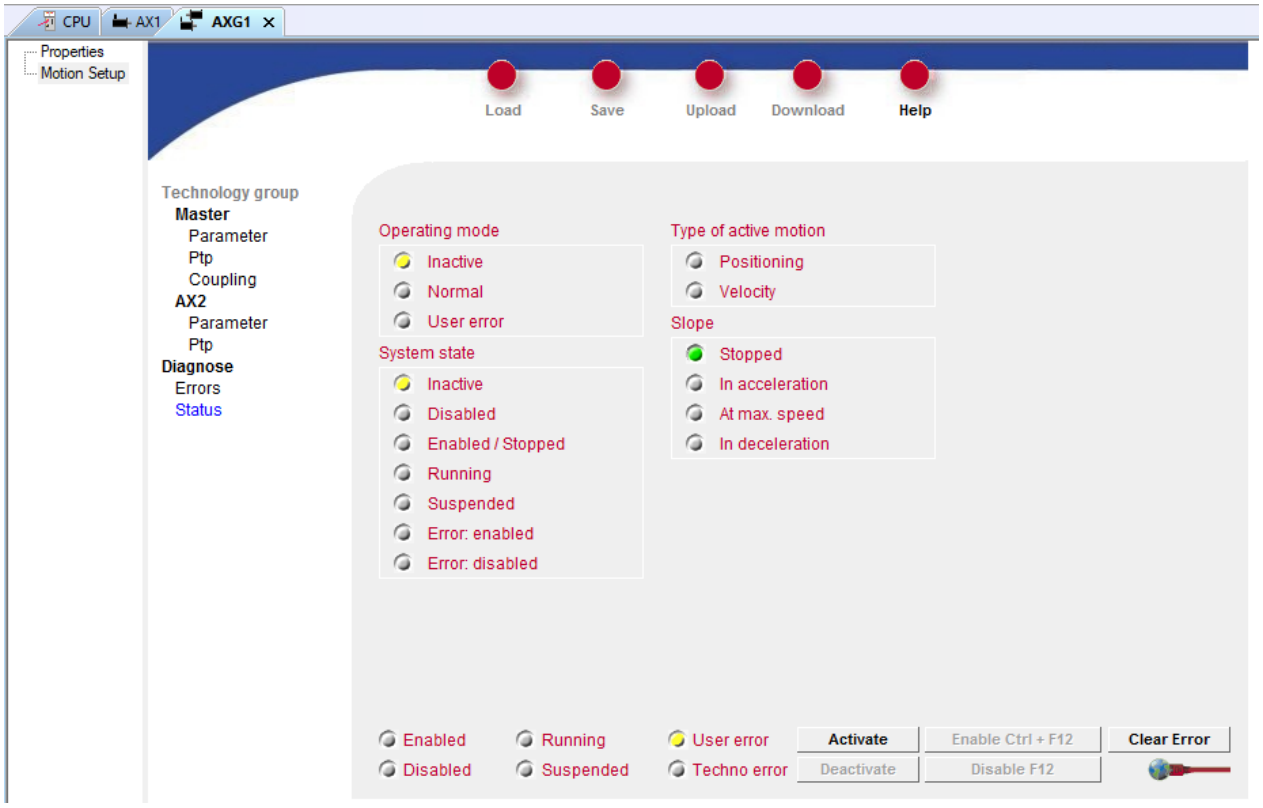
This overview lists the current errors, warnings and messages of the group and its member objects.
The "Clear Error" button can be used to acknowledge pending errors. If "Clear Error" is applied for the group, this also clears the errors of its members.



3.3.2 Status Overview - Group

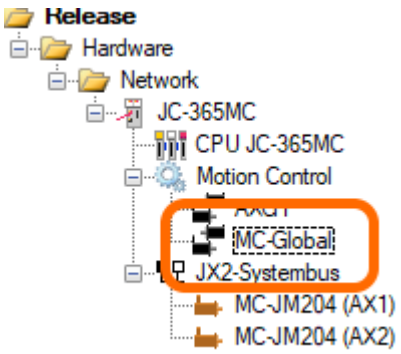
As with the status overview of the single axis, the current states of the axis group are displayed here.
There are 4 categories available:

- Operating mode
- Operating state
- Type of active motion
- Ramp (slope) status



3.4 Motion Setup – MC-Global

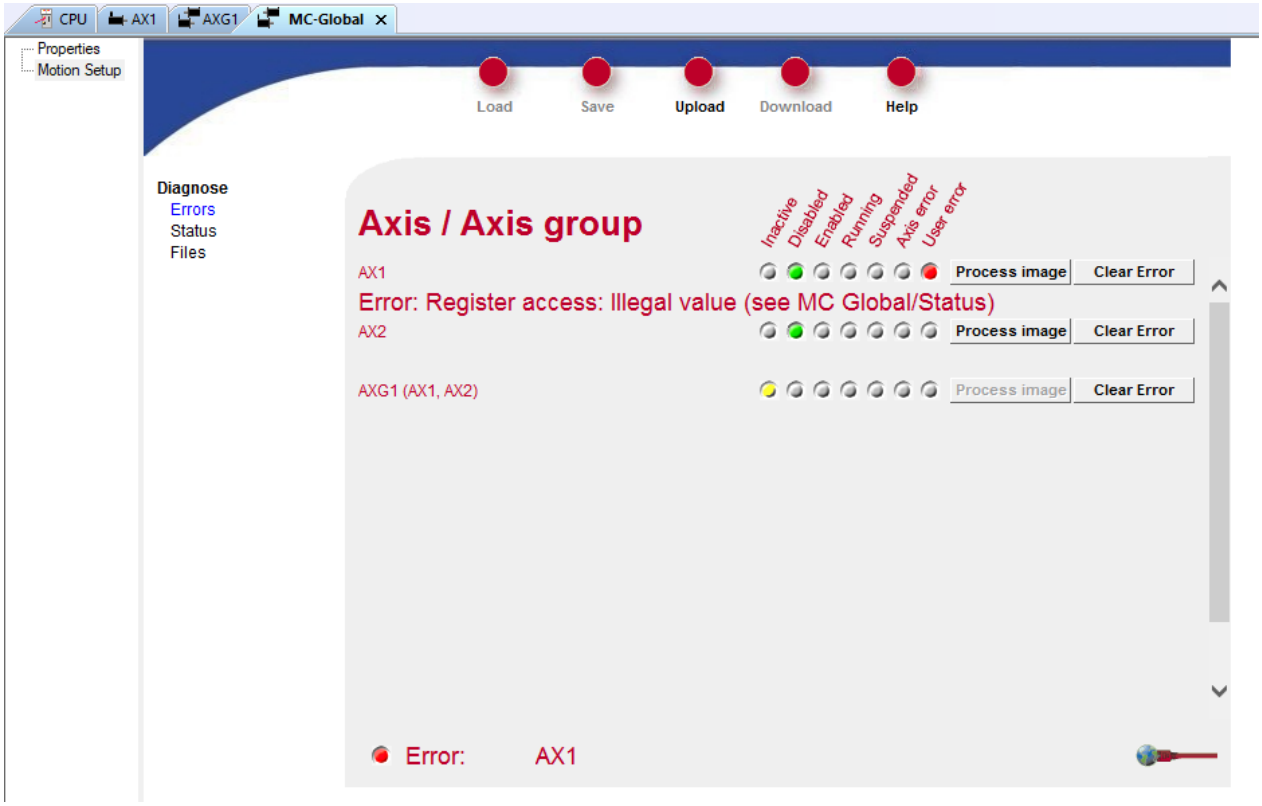
MC-Global provides an overview of the entire MCX system.



The "MC-Global" node is created in the hardware tree as soon as an MCX object is created.

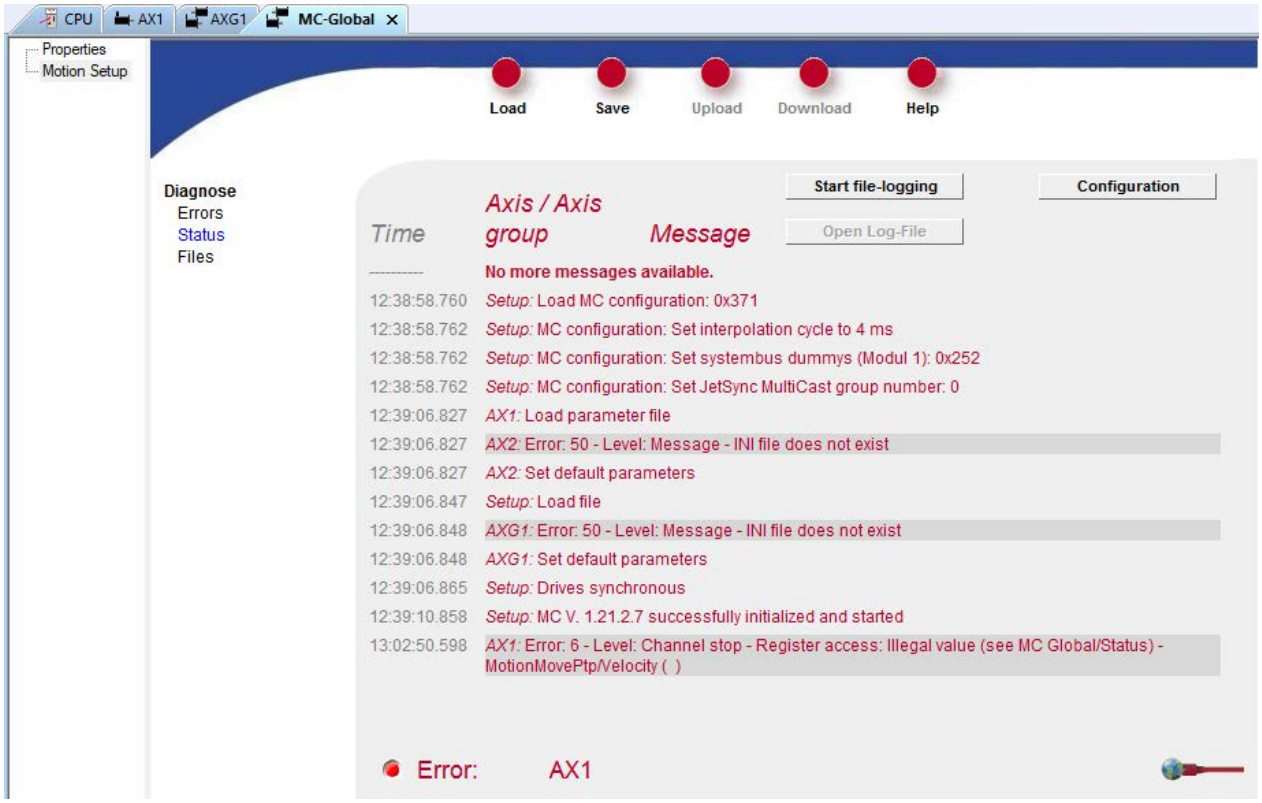
3.4.1 Errors

All axis objects created in the Hardware Manager are listed in this view.
The operating status of each object is displayed.
Errors and warnings are listed for each object.
The "Clear Error" button can be used to acknowledge pending errors. If "Clear Error" is applied for the group, this also clears the errors of its members.
A process image can be generated for later analysis. A "*_Tmp99.ini" file is created in the file system of the controller with the currently active axis parameters.



3.4.1.1 Status

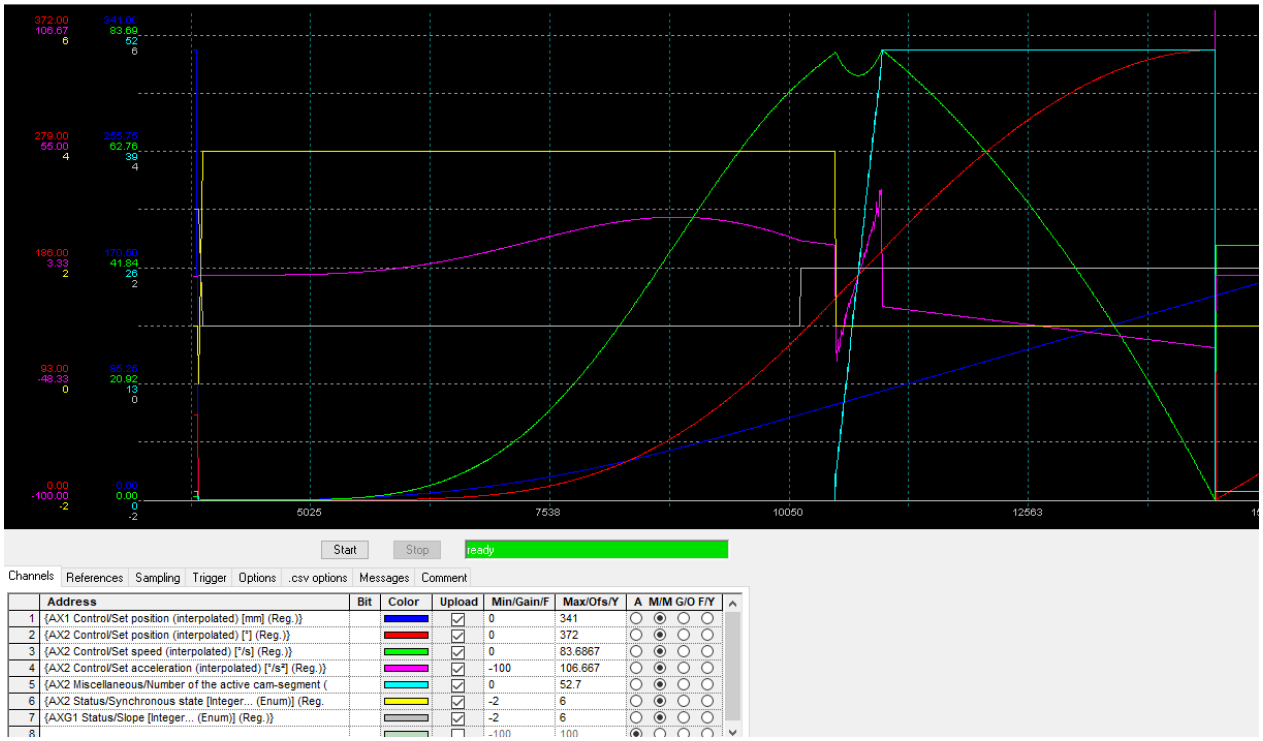
The MCX stores the accesses or commands in a ring buffer.
This allows you to trace the time history of MCX events.



This example shows the boot history, which ends with "…successfully initialized and started". However, you can also see that an invalid value was entered for axis 1 when MovePtp was called. Although this message does not show for which parameter which invalid value was entered, but this message can be used to further investigate in the application program.
In this way, it is possible to trace which command caused an error or in which phase errors occurred.

3.5 Oscilloscope

The JetSym oscilloscope lets you not only record positions and speeds.
MCX provides many other parameters that may be of interest for analyzing a sequence. Furthermore, registers/variables from the application program can also be recorded.
Thanks to these functions, the oscilloscope is not only a tool for commissioning axes, but also for debugging and visualizing application program sequences.



In this example, in addition to the setpoint positions of the master and follower axes, the setpoint velocity and the setpoint acceleration are also shown.
In addition, the curve characteristic can also be checked by displaying the segment number of a cam profile.
The behavior during the different coupling phases is checked by recording the coupling status.
With the help of the ramp status, it can be seen that the master axis has only reached the target velocity shortly from the synchronization point.

4 Application Program - Motion API

4.1 MCX Boot Status:

The MCX R59997904 system register shows whether the boot process was successful. A status value of "***21***" signals that the MCX kernel has started and was able to create and initialize the objects (axes, group).
It is sufficient to check this value when starting the application, as the value does not change during runtime.
If the value is not equal to "***21***", the MCX could not start successfully.
The application program can generate a user message here.
This error cannot be acknowledged. User intervention is required.
Example: The specified hardware does not match the connected hardware.

- Check the set hardware configuration and, if necessary, transfer it to the controller again.
- Are the connected drives powered up?
- Are the addresses correct?
- Check the connection cables.
- …

4.2 MCX Errors in the Application Program

4.2.1 Collective Errors

If an error occurs for one or more MC objects, this error is displayed as a group error. The displayed errors are to be regarded as equal.

4.2.2 System Error Registers

System errors are displayed in the higher-level error registers of the controller:
Error register 1 R200008.Bit7 indicates that an error is present in other error registers.
Error register 3 R200010.Bit1 indicates that there is an error in the Motion Control.
To detect Motion Control errors, it is sufficient to monitor only R200010.Bit1. For cascaded monitoring it is sufficient to monitor whether error register 1 is not equal to 0. In some cases, a corresponding error response is already triggered here and the analysis is then continued in the following. If a deviation is detected, the additional error registers are only checked when bit 7 is set in this register. If then error register3.bit1 is found to be set, the error response or the analysis of the errors is continued.

4.2.3 Number of MCX Objects with Errors

The number of MCX objects with errors can be determined via the Motion API command *mcMgr.FaultyObjectCount* . If a value > 0 is returned here, at least one Motion Control object has an error. It is then possible to proceed with the error response or analysis.

4.2.4 Checking Individual Motion Control Objects

The Motion API provides a corresponding diagnostic object for each Motion Control object created.

Motion API 1.0	Motion API 2.0	Error location
MCAxis.	*MCAxis.*	
Diagnostics.IsErrorPending	Diagnostics.IsErrorPending	Axis object
Diagnostics.Software.IsErrorPending()		MCX
Diagnostics.Drive.IsErrorPending()	Drive.Diagnostics.IsErrorPending	Drive
Diagnostics.Drive.IsErrorPending(ErrorNr)		Specific drive *1)

Motion API 1.0	Motion API 2.0	Error location
MCGeo.	*MCGeo.*	
Diagnostics.IsErrorPending	Diagnostics.IsErrorPending	Path Group
MCTechno.	*MCTechno.*	
Diagnostics.IsErrorPending	Diagnostics.IsErrorPending	Technology group

1) MCAxis.Diagnostics.Drive.IsErrorPending(ErrorNr): There is exactly this error at this drive. The ErrorNr corresponds exactly to the error number of the JetMove 100/200. See also the user information on the JetMove 2xx: jetmove_2xx_at_jetcontrol_bi_2114_userinformation.pdf

Example:

```
// an error occurred on MyAxis
when MyAxis.Diagnostics.IsErrorPending continue;
// execute error handling and create an alarm
```

```
// check which axis has an error
if MyAxis1.Diagnostics.IsErrorPending then
// execute error handling and create an alarm
end_if;
if MyAxis2.Diagnostics.IsErrorPending then
//….
else
//….
end_if;
```

4.2.5 Error Evaluation

4.2.5.1 Motion Control Errors

For the Motion Control objects created, errors are stored in a buffer until they are cleared. The Motion API provides the functions to read out this buffer.

MotionApi 1.0
Pending errors for all object types are read back in the same way.

MotionApi 2.0
For axis objects, pending errors are read back directly in the diagnostic object, while errors of group objects are read back in the same way as in MotionApi1.0.

Motion API 1.0	Motion API 2.0	Description
*MCAxis.**Diagnostics**.**Software***	*MCAxis.**Diagnostics**.*	
ErrorCount	ErrorCount	Number of error entries in the buffer
GetErrorCode(BufferIndex)	GetErrorCode(BufferIndex)	Error entry
MCGeo.Diagnostics.Software		
ErrorCount		Number of error entries in the buffer
GetErrorCode(BufferIndex)		
MCTechno.Diagnostics.Software		
ErrorCount		Number of error entries in the buffer
GetErrorCode(BufferIndex)		

ErrorCount: Number of error entries in the buffer
GetErrorCode(BufferIndex): Returns pending errors. Via BufferIndex all entries can be read out one after the other. If the function parameter remains empty, the first entry in the buffer is always returned.

Example:
When in this Motion API 2.0 example an MCX error occurs, the MC Manager object "mcMgr" is used to systematically read out the pending errors from all objects with an error.

```
#include ".\GeneralFunctions.stxp"

type
   SystemExtErrorRegsiter2:   bits(
      MCX = 1
   );
end_type;

function superviseAnyError()
var
   nListIndex:                              int32;
   nErrorIndex:                             int32;
   pDiagnostics:      pointer to     MCDiagnosticsSoftware;
end_var;

   if mcMgr.FaultyObjectCount > 0 then
      for nListIndex := 0 to mcMgr.FaultyObjectCount by 1 do
         pDiagnostics := 0;
         case mcMgr.FaultyObject[nListIndex].Info.ClassID of
         MCClassIDs.Axis:
            pDiagnostics :=  mcMgr.AxisList[nListIndex].Diagnostics;
```

```
         break;
         MCClassIDs.Techno:
            pDiagnostics :=  mcMgr.TechnoList[nListIndex].Diagnostics.Software;
         break;
         MCClassIDs.Geo:
            pDiagnostics :=  mcMgr.GeoList[nListIndex].Diagnostics.Software;
         break;
         end_case;

         if pDiagnostics != 0 then
            for nErrorIndex := 0 to mcMgr.AxisList[nListIndex].Diagnostics.ErrorCount by 1 do
               // Error Evaluation
               trace(mcMgr.FaultyObject[nListIndex].Info.Name + ': '+ IntTo-
Str(pDiagnostics.GetErrorCode(nErrorIndex)) + ';$n');
            end_for;
         end_if;
      end_for;
   end_if;
end_function;

var
   g SystemErrorExt2:         SystemExtErrorRegsiter2 at %vl 200010;
   g_bClearError:                    bool;
end_var;

// Task supervises if any MCX-error occured
task supervision autorun

   when g SystemErrorExt2.MCX continue;
      superviseAnyError();
      g_bClearError := false;
      when g_bClearError continue;
      resetTechnoAll();
      resetAxisAll();
      g_SystemErrorExt2.MCX := false;

end_task;
```

4.2.5.2 Servoamplifier Errors

Motion API 1.0 with JetMove2xx

You can evaluate existing errors either by directly querying the error or by reading out the superimposed error registers. In these registers, errors are bit-coded.
MCAxis.Diagnostics.Drive.IsErrorPending(ErrorNr): If true, this error is present.
MCAxis.Diagnostic.Drive. ErrorFlags1: Bit-coded errors 00 to 31 on the JetMove2xx.
MCAxis.Diagnostic.Drive. ErrorFlags2: Bit-coded errors 32 to 43 on the JetMove2xx.
The bit-wise evaluation can be done e.g. in a counting loop. The meaning of the individual error bits is described in the User Information on the JetMove 2xx:
jetmove_2xx_at_jetcontrol_bi_xxxx_user_information.pdf

MotionApi 2.0 with EtherCAT drives

MCAxis.Drive.Diagnostics.IsErrorPending: If true, an error is pending.
MCAxis.Drive.Diagnostics.ActualError: Return value of the pending error. Refer to the relevant user manual for the error numbers.

Motion API 1.0	Motion API 2.0	Description
*MCAxis.**Diagnostics.Drive***	*MCAxis.**Drive.Diagnostics***.	
IsErrorPending(ErrorNr)	IsErrorPending	Number of error entries in the buffer
ErrorFlags1	*ActualError*	Error entry
ErrorFlags2		Error entry

4.2.6 Warnings and Messages: Detection and Evaluation

4.2.6.1 Motion Control Messages

In addition to error messages, the Motion Control also generates general messages. This is information that may be of interest for reviewing processes.
If, for example, a cam disc is to be activated at a certain master axis position and another cam disc is activated before this point is reached, the message is generated that a pending cam activation is aborted.

Motion API 1.0	Motion API 2.0	Description
*MCAxis.**Diagnostics.***	*MCAxis.**Diagnostics.***	
IsMessagePending		There is a message.
*MCAxis.**Diagnostics.Software***	*MCAxis.**Diagnostics.***	
IsMessagePending		There is a message.
MessageCount		Number of messages in the buffer.
GetMessageCode(BufferIndex)		Message entry
MCTechno.**Diagnostics**.		
MCGeo.**Diagnostics**.		
IsMessagePending		There is a message.
.**Software.**		
IsMessagePending		There is a message.
MessageCount		Number of messages in the buffer.
GetMessageCode(BufferIndex)		Message entry

<MCObject>.Diagnostics.IsMessagePending and *<MCObject>.Diagnostics.Software.IsMessagePending* are identical properties of the diagnostics object.

4.2.6.2 Drive Messages

Motion API 1.0	Motion API 2.0	Description
	*MCAxis.Drive.**Diagnostics.***	
-	IsMessagePending	There is a message.

4.2.6.3 Drive Warnings

In the MCX system, warnings are generated only by the drive amplifier.
Motion API 1.0 lets you read out not only existing warnings, but also the contents of the warning register on the JM-2xx. The meaning of the individual warning bits is described in the User Information on the JetMove 2xx:
jetmove_2xx_at_jetcontrol_bi_xxxx_user_information.pdf
Motion API 2.0 can only detect that warnings are present. If you want to read out the warning, then this must be done by means of a CANopen object. These objects are described in the manual of the respective drive.

Motion API 1.0	Motion API 2.0	Description
*MCAxis.**Diagnostics.***		
IsWarningPending		There is a warning.
*MCAxis.**Diagnostics.Drive***	*MCAxis.**Drive.Diagnostics***	
IsWarningPending	IsWarningPending	There is a warning.
WarningFlags		JM-2xx warning register

Jetter automation

Jetter AG
Graeterstrasse 2
71642 Ludwigsburg
Germany
www.jetter.de

E-mail info@jetter.de
Phone +49 7141 2550-0

We automate your success.