Allen-Bradley

ControlLogix Multi-Vendor Interface Module DH-485 API

1756-MVI

User Manual

Rockwell Automation

Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI-1.1, *Safety Guidelines for the Application, Installation and Maintenance of Solid-State Control* (available from your local Allen-Bradley office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

Throughout this manual we use notes to make you aware of safety considerations:

ATTENTION

Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss

Attention statements help you to:

- identify a hazard
- avoid a hazard
- recognize the consequences

IMPORTANT

Identifies information that is critical for successful application and understanding of the product.

European Communities (EC) Directive Compliance

If this product has the CE mark it is approved for installation within the European Union and EEA regions. It has been designed and tested to meet the following directives.

EMC Directive

This product is tested to meet the Council Directive 89/336/EC Electromagnetic Compatibility (EMC) by applying the following standards, in whole or in part, documented in a technical construction file:

- EN 50081-2 EMC — Generic Emission Standard, Part 2 — Industrial Environment
- EN 50082-2 EMC — Generic Immunity Standard, Part 2 — Industrial Environment

This product is intended for use in an industrial environment.

Low Voltage Directive

This product is tested to meet Council Directive 73/23/EEC Low Voltage, by applying the safety requirements of EN 61131-2 Programmable Controllers, Part 2 - Equipment Requirements and Tests. For specific information required by EN 61131-2, see the appropriate sections in this publication, as well as the Allen-Bradley publication Industrial Automation Wiring and Grounding Guidelines For Noise Immunity, publication 1770-4.1.

This equipment is classified as open equipment and must be mounted in an enclosure during operation to provide safety protection.

About This User Manual

Introduction

This user manual provides information needed to develop application programs for the 1756-MVI ControlLogix Multi-Vendor Interface Module using the DH-485 API (Application Programming Interface).

This user manual describes the available software DH-485 API libraries and tools, programming information, and example code.

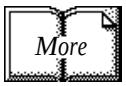
Audience

This user manual is intended for control engineers and technicians who are installing, programming, and maintaining a control system that includes a 1756-MVI module.

We assume that you:

- are familiar with software development in the 16-bit DOS environment using the C programming language.
- are familiar with Allen-Bradley programmable controllers and the ControlLogix platform.

References



For additional information refer to the following publications:

- ControlLogix 1756-MVI Multi-Vendor Interface Module Installation Instructions, publication number1756-1N001A-US-P
- ControlLogix 1756-MVI Multi-Vendor Interface Module Programming Reference Manual, publication number1756-RM004A-EN-P
- General Software Embedded DOS 6-XL Developer's Guide 1.2
- Introduction to ControlLogix Module Development, CID#X1557

Rockwell Automation Support

Rockwell Automation offers support services worldwide, with over 75 sales/support offices, 512 authorized distributors, and 260 authorized systems integrators located throughout the United States alone, plus Rockwell Automation representatives in every major country in the world.

Local Product Support

Contact your local Rockwell Automation representative for:

- sales and order support
- product technical training
- warranty support
- support service agreements

Technical Product Assistance

If you need to contact Rockwell Automation for technical assistance, call your local Rockwell Automation representative, or call Rockwell directly at: 1 440 646-6800.

For presales support, call 1 440 646-3NET.

You can obtain technical assistance online from the following Rockwell Automation WEB sites:

- www.ab.com/mem/technotes/kbhome.html (knowledge base)
- www.ab.com/networks/eds (electronic data sheets)

Your Questions or Comments about This Manual

If you find a problem with this manual, please notify us of it on the enclosed Publication Problem Report (at the back of this manual).

If you have any suggestions about how we can make this manual more useful to you, please contact us at the following address:

Rockwell Automation, Allen-Bradley Company, Inc.
Control and Information Group
Technical Communication
1 Allen-Bradley Drive
Mayfield Heights, OH 44124-6118

MVI DH-485 API

Chapter 1

Index

MVI DH-485 API

The DH-485 API is one component of the 1756-MVI API Suite. The DH-485 API allows applications to communicate with foreign devices over the serial ports in the RS-485 mode using the DH-485 Link Layer.

The DH-485 API provides a common applications interface for all of the modules in the MVI family. This common interface allows application portability between modules in the family.

What This Chapter Contains

The following table identifies what this chapter contains and where to find specific information.

DH-485 API Files

Table 1.A lists the supplied DH-485 API file names. These files should be copied to a convenient directory on the computer where the application is to be developed. These files need not be present on the module when executing the application.

Table 1.A Serial API File Names

FileName	Description
dh485api.h	Include file
dh485api.lib	Library (16-bit OMF format)

DH-485 Serial Data Transfer

The DH-485 API communicates with remote DH-485 devices via standard UART hardware. The API acts as a high level interface that hides the details of the DH-485 protocol from the application programmer.

The primary purpose of the API is to allow data to be transferred between the module and a remote DH-485 device. The application needs to be programmed to implement the specific requirements the remote device, and the data can then be processed by the application and transferred to the control processor.

Note: The 1756-MVI hardware only supports RS-485 communication on ports 2 and 3. The MVI jumpers must be correctly set to RS-485 mode. The DH-485 API uses the MVI Serial Port API to control the serial port hardware. The application should include the Serial port API library file MVISPAPI.LIB when being linked. See the 1756-MVI Developers guide for details.

DH-485 API Functions

This section provides detailed programming information for each of the API library functions. The calling convention for each API function is shown in C format.

The API library routines are categorized according to functionality as shown in Table 1.B.

Table 1.B - DH-485 API Functions

Function Category	Function Name	Description
Initialization	MVIdh_Open	Initializes access to a DH485 serial port.
	MVIdh_Close	Terminates access to the DH485 serial port.
Port Status	MVIdh_GetANTable	Get the active node table.
	MVIdh_GetCommStatus	Get the DH-485 communication status.
	MVIdh_GetLedState	Get the DH-485 LED state.
Communications	MVIdh_GetDataFromCIF	Retrieve data from the CIF data buffer
	MVIdh_PutDataToCIF	Save data to the CIF data buffer
	MVIdh_CheckCIFRdStatus	Checks the read status of the CIF data buffer
	MVIdh_CheckCIFWrStatus	Checks the write status of the CIF data buffer
	MVIdh_WriteRemoteCIFFile	Write data to a remote device's CIF file
	MVIdh_ReadRemoteCIFFile	Read data from a remote device's CIF file
	MVIdh_WriteRemoteDataFile	Write data to a remote device's data file
	MVIdh_ReadRemoteDataFile	Read data from a remote device's data file
Miscellaneous	MVIdh_GetVersionInfo	Get the DH-485 API version information
	MVIdh_ErrorString	Get a text description for an error code

Initialization Functions

MVIdh_Open

Syntax:

int MVIdh_Open(MVIHANDLE *handle, DH485CONFIG *dh485cfg);

Parameters:

handle	Pointer to variable of type MVIHANDLE
dh485cfg	Pointer to DH485CONFIG structure containing the DH485 configuration data

Description:

MVIdh_Open acquires access to the API and sets *handle* to a unique ID that the application uses in subsequent functions. This function also acquires to the specified serial port and allocates any resources needed by the API. This function must be called before any of the other API functions can be used.

dh485cfg specifies which *comport* is to be opened, the *baudrat*e, *node* number, and the *mode*. The valid values for *comport* on the 1756-MVI module are MVI_COM2 (corresponds to PRT2) and MVI_COM3 (corresponds to PRT3). Valid Values for *baudrate* are MVI_BAUD_1200, MVI_BAUD_2400, MVI_BAUD_9600, and MVI_BAUD19200. The node can be set to any number from 0 to 31. The API supports two DH-485 mode values, MVI_MODE_SLAVE and MVI_MODE_MASTER.

```
typedef struct tagDH485CONFIG
{
int   comport;              /* COM2, COM3 *
      BYTE    baudrate;     /* BAUD_1200 - BAUD_19200 */
      BYTE    node;         /* Valid nodes are 0 - 31 */
      BYTE    mode;         /* 0 = slave, 2 = master */
} DH485CONFIG;
```

IMPORTANT Once the API has been opened, MVIdh_Close should always be called before exiting the application.

Return Value:

MVI_SUCCESS	port was opened successfully
MVI_ERR_REOPEN	port is already open
MVI_ERR_NODEVICE	UART not found on port

Note: MVI_ERR_NODEVICE will be returned if the port is not supported by the module.

MVIdh_Open

Example:

```
DH485CONFIG dhcfg;

dhcfg.comport = MVI_COM2;
dhcfg.baudrate = MVI_BAUD_9600;
dhcfg.node = 5;
dhcfg.mode = MVI_MODE_MASTER;

if ( MVIdh_Open(handle, &dhcfg) != MVI_SUCCESS) {
    printf("Open failed!\n");
} else {
    printf("Open succeeded\n");
}
```

See Also:

MVIdh_Close

MVIdh_Close

Syntax:

int MVIdh_Close(MVIHANDLE handle);

Parameters:

handle Handle returned by previous call to MVIdh_Open

Description:

This function is used by an application to release control of the DH-485 API. *handle* must be a valid handle returned from MVIdh_Open.

IMPORTANT Once the DH-485 API has been opened, this function should always be called before exiting the application.

Return Value:

MVI_SUCCESS	API was closed successfully
MVI_ERR_NOACCESS	*handle* does not have access

Example:

```
MVIHANDLE handle;
MVIdh_Close(handle);
```

See Also:

MVIsp_Open

Port Status Functions

MVIdh_GetANTable

Syntax:

int MVIdh_GetANTable(MVIHANDLE handle, DWORD *ANTable);

Parameters:

handle	Handle returned by previous call to MVIdh_Open
ANTable	Pointer to variable that will receive the Active node table

Description:

This function is used to retrieves a copy of the DH-485 network's active node table. *handle* must be a valid handle returned from MVIdh_Open.

ANTable is a pointer to a double word. When this function returns, each bit of *ANTable* will be set to 1 if the corresponding node number is active on the DH-485 network.

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access

Example:

```
DWORD ANTable;
if (MVIdh_GetANTable(handle, &ANTable) != MVI_SUCCESS) {
    printf("Get ANT failed!\n");
}
```

MVIdh_GetCommStatus

Syntax:

int MVIdh_GetCommStatus(MVIHANDLE handle, BYTE *bStatus);

Parameters:

handle	Handle returned by previous call to MVIdh_Open
bStatus	Pointer to a byte that is set to 1 if online and 0 if offline

Description:

This function is used to query the state of the DH-485 port. *handle* must be a valid handle returned from MVIdh_Open.

bStatus is a pointer to a byte. When this function returns, *bStatus* will be set to MVI_COMM_STATUS_ON if the port is online, or MVI_COMM_STATUS_OFF if the port is offline.

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access

Example:

```
MVIHANDLE   handle;
BYTE        status;

MVIdh_GetCommStatus(handle, &status);

if (status == MVI_COMM_STATUS_ON)
        // Communication port is online

else
        // Communication port is offline
```

MVIdh_GetLedState

Syntax:

int MVIdh_GetLedState(MVIHANDLE handle, BYTE *bState);

Parameters:

handle	Handle returned by previous call to MVIdh_Open
bState	Pointer to a byte that is set to 1 if DH-485 LED state is on, or 0 if off

Description:

This function is used to query the state of the DH-485 port's LED. An application can use the state returned to turn an LED on or off. *handle* must be a valid handle returned from MVIdh_Open.

bState is a pointer to a byte. When this function returns, *bState* will be set to MVI_LED_STATE_ON if the DH-485 state machine is indicating the LED should be on, or MVI_LED_STATE_OFF if the LED should be off.

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access

Example:

```
MVIHANDLE   handle;
BYTE        state;

MVIdh_GetLEDState(handle, &state);
if (status == MVI_LED_STATE_ON)
// Turn user LED 1 on

else
        // Turn user LED 1 off
```

Communication Functions

MVIdh_GetDataFromCIF

Syntax:

```
int    MVIdh_GetDataFromCIF(
            MVIHANDLE handle,
            WORD offset,
            WORD datasize,
            BYTE *dataBuf );
```

Parameters:

handle	Handle returned by previous call to MVIdh_Open
offset	Offset in bytes from which to start getting data from the CIF
dataSize	Number of bytes to read from the CIF
dataBuf	Pointer to buffer to receive the data read from the CIF

Description:

This function is used to transfer *dataSize* bytes of data starting at offset from the DH-485 Common Interface File to an application buffer pointed to by *dataBuf*. *handle* must be a valid handle returned from MVIdh_Open.

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access
MVI_ERR_BADPARAM	*offset* or *dataSize* is invalid

Example:

```
MVIHANDLE  Handle;
BYTE       buffer[128];

// Write 128 bytes to the CIF data buffer

if( MVI_SUCCESS != MVIdh_GetDataFromCIF(Handle, 0, 128, buffer ))
{
      printf( "Get Data from CIF Failed\n");
}
```

See Also:

MVIdh_PutDataToCIF

MVIdh_PutDataToCIF

Syntax:

```
int     MVIdh_PutDataToCIF(
                MVIHANDLE handle,
                BYTE *dataBuf,
                WORD offset,
                WORD dataSize );
```

Parameters:

handle	Handle returned by previous call to MVIdh_Open
dataBuf	Pointer to buffer from which data is copied to the CIF
offset	Offset in bytes from which to start writing data into the CIF
dataSize	Number of bytes to write to the CIF

Description:

This function is used to transfer *dataSize* bytes of data from an application buffer pointed to by *dataBuf* to the DH-485 Common Interface File starting at *offset*. *handle* must be a valid handle returned from MVIdh_Open.

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access
MVI_ERR_BADPARAM	*offset* or *dataSize* is invalid

Example:

```
MVIHANDLE   Handle;
BYTE        buffer[128];

// Write 128 bytes to the CIF data buffer
if( MVI_SUCCESS != MVIdh_PutDataToCIF(Handle, buffer, 0, 128 ))
{
      printf( "Put Data to CIF Failed\n");
}
```

See Also:

MVIdh_GetDataFromCIF

MVIdh_CheckCIFRdStatus

Syntax:

int MVIdh_CheckCIFRdStatus(MVIHANDLE handle, BYTE *bStatus);

Parameters:

handle	Handle returned by previous call to MVIdh_Open
bStatus	Read status of the CIF buffer, 0 if not read since it was last checked

Description:

This function is used to check the read status of the DH-485 Common Interface File. *bStatus* returns MVI_CIF_ACCESSED if the CIF has been read since it was last checked and MVI_CIF_NOTACCESSED if it has not been read. *handle* must be a valid handle returned from MVIdh_Open.

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access

Example:

```
MVIHANDLE   Handle;
int         timeout;

timeout = 1000;
// Wait for CIF file to be read
while (--timeout)
{
   if ( MVI_SUCCESS != MVIdh_CheckCIFRdStatus( Handle, &bStatus ) )
   {
      printf( Check of CIF read status failed \n");
      Break;
   }
   if ( bStatus == MVI_CIF_ACCESSED )
      break;
}
```

See Also:

MVIdh_CheckCIFWrStatus

MVIdh_CheckCIFWrStatus

Syntax:

int MVIdh_CheckCIFWrStatus(MVIHANDLE handle, BYTE *bStatus);

Parameters:

handle	Handle returned by previous call to MVIdh_Open
bStatus	Write status of the CIF buffer, 0 if not written since it was last checked

Description:

This function is used to check the write status of the DH-485 Common Interface File. *bStatus* returns MVI_CIF_ACCESSED if the CIF has been written since it was last checked and MVI_CIF_NOTACCESSED if it has not been written. *handle* must be a valid handle returned from MVIdh_Open.

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access

Example:

```
MVIHANDLE   Handle;
int         timeout;

timeout = 1000;

// Wait for CIF file to be written
while (--timeout)
{
    if ( MVI_SUCCESS != MVIdh_CheckCIFWrStatus( Handle, &bStatus ) )
    {
        printf( Check of CIF write status failed \n");
        Break;
    }
    if ( bStatus == MVI_CIF_ACCESSED )
        break;
}
```

See Also:

MVIdh_CheckCIFRdStatus

MVIdh_WriteRemoteCIFFile

Syntax:

```
int    MVIdh_WriteRemoteCIFFile(
            MVIHANDLE handle,
            BYTE *dataBuf,
            WORD node,
            WORD offset,
            WORD dataSize,
            WORD timeout );
```

Parameters:

handle	Handle returned by previous call to MVIdh_Open
dataBuf	Pointer to buffer from which data is copied to the remote device's CIF
node	Node number of remote device to access
offset	Offset in bytes in the remote device's CIF in which to write data
dataSize	Number of bytes to write to the remote device's CIF
timeout	Time to wait for remote device to respond in 100's of milliseconds

Description:

This function is used to write data to the DH-485 Common Interface file on a remote device at *node* address. *dataSize* bytes will be copied from the application buffer pointed to by *dataBuf* to the remote node's CIF data file starting at *offset*. If a response is not received in *timeout*, the function aborts and returns a timeout error. *handle* must be a valid handle returned from MVIdh_Open.

MVIdh_WriteRemoteCIFFile

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access
MVI_ERR_BADPARAM	Invalid *node*, *offset*, *dataSize*, or *timeout*
MVI_ERR_NOT_MASTER	This function can only be executed in master mode
MVI_ERR_TXCMD_BUSY	Transmitter is already executing a command
MVI_ERR_RM_MSGTIMEOUT	Remote device did not respond in *timeout* period
MVI_ERR_MEM_ALLOC	Unable to allocate memory for request
MVI_ERR_ILL_CMD_FMT	Target node, illegal command or format
MVI_ERR_ADDRESS_PROBLEM	Target node out of memory, file or rung doe snot exist

Example:

```
MVIHANDLE   Handle;
BYTE        databuf[20];

// Write data to remote CIF

if ( MVI_SUCCESS != MVIdh_WriteRemoteCIFFile(Handle, &databuf[0], 15, 0,
20, 30) )
{
      printf( Write to remote CIF failed \n");
}
```

See Also:

MVIdh_ReadRemoteCIFFile

MVIdh_ReadRemoteCIFFile

Syntax:

```
int    MVIdh_ReadRemoteCIFFile(
            MVIHANDLE handle,
            WORD node,
            WORD offset,
            WORD dataSize,
            BYTE *dataBuf,
            WORD timeout );
```

Parameters:

handle	Handle returned by previous call to MVIdh_Open
node	Node number of remote device to access
offset	Offset in bytes in the remote device's CIF from which to read data
dataSize	Number of bytes to read from the remote device's CIF
dataBuf	Pointer to buffer into which data is copied from the remote device's CIF
timeout	Time to wait for remote device to respond in 100's of milliseconds

Description:

This function is used to read data from the DH-485 Common Interface file on a remote device at *node* address. *dataSize* bytes will be copied to the application buffer pointed to by *dataBuf* from the remote node's CIF data file starting at offset. If a response is not received in *timeout,* the function aborts and returns a timeout error. *handle* must be a valid handle returned from MVIdh_Open.

MVIdh_ReadRemoteCIFFile

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access
MVI_ERR_BADPARAM	Invalid *node*, *offset*, *dataSize*, or *timeout*
MVI_ERR_NOT_MASTER	This function can only be executed in master mode
MVI_ERR_TXCMD_BUSY	Transmitter is already executing a command
MVI_ERR_RM_MSGTIMEOUT	Remote device did not respond in timeout period
MVI_ERR_MEM_ALLOC	Unable to allocate memory for request
MVI_ERR_ILL_CMD_FMT	Target node, illegal command or format
MVI_ERR_ADDRESS_PROBLEM	Target node out of memory, file or rung does not exist

Example:

```
MVIHANDLE   Handle;
BYTE        databuf[20];

// Read remote CIF data into buffer

if ( MVI_SUCCESS != MVIdh_ReadRemoteCIFFile(Handle, 15, 0, 20, &databuf[0],
30) )
{
      printf( Read of remote CIF data failed \n");
}
```

See Also:

MVIdh_WriteRemoteCIFFile

MVIdh_WriteRemoteDataFile

Syntax:

```
int     MVIdh_WriteRemoteDataFile(
                MVIHANDLE handle,
                BYTE *dataBuf,
                BYTE node,
                BYTE numElements,
                WORD fileNum,
                BYTE fileType,
                WORD Element,
                WORD timeout );
```

Parameters:

handle	Handle returned by previous call to MVIdh_Open
dataBuf	Pointer to buffer from which data is copied to the remote device's data file
node	Node number of remote device to access
numElements	Number of data elements to write to the remote device's data file
fileNum	Number of remote device's data file to access
fileType	Type of remote device's data file to access
Element	Element number of remote device's data file to start writing to
timeout	Time to wait for remote device to respond in 100's of milliseconds

MVIdh_WriteRemoteDataFile

Description:

This function copies data from *dataBuf* to a remote device at *node* address. *numElements* data elements will be copied from the application buffer to a remote node's data file. If a response is not received in *timeout*, the function aborts and returns a timeout error. *handle* must be a valid handle returned from MVIdh_Open.

fileNum is the data file number to be written to on the remote device.

fileType is the type of file being accessed. Valid types are listed in table 1.C.

Element is the offset into the data file to start writing the data. The number of bytes per element is dependent on the file type and is listed in table 1.C.

Table 1.C - Valid File Types

File Type	Number of bytes per element
MVI_FILETYPE_STATUS	2
MVI_FILETYPE_BIT	2
MVI_FILETYPE_TIMER	6
MVI_FILETYPE_COUNTER	6
MVI_FILETYPE_CONTROL	6
MVI_FILETYPE_INTEGER	2

MVIdh_WriteRemoteDataFile

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access
MVI_ERR_BADPARAM	Invalid *node*, *offset*, *dataSize*, or *timeout*
MVI_ERR_NOT_MASTER	This function can only be executed in master mode
MVI_ERR_TXCMD_BUSY	Transmitter is already executing a command
MVI_ERR_RM_MSGTIMEOUT	Remote device did not respond in timeout period
MVI_ERR_MEM_ALLOC	Unable to allocate memory for request
MVI_ERR_ILL_CMD_FMT	Target node, illegal command or format
MVI_ERR_ADDRESS_PROBLEM	Target node out of memory, file or rung doesn't exist
MVI_ERR_CMD_EXECUTION	Target node command can't be executed
MVI_ERR_FILE_OPEN	Target node file open by another node
MVI_ERR_PROGRAM_OWNED	Target node program owned by another node

Example:

```
MVIHANDLE   Handle;
BYTE        dataBuf[20];

// Write Remote Data file

if ( MVI_SUCCESS != MVIdh_WriteRemoteDataFile(Handle, 15, 10, 9,
    MVI_FILETYPE_INTEGER, 0, &databuf[0], 30) )
{
       printf( Read of remote data file failed \n");
}
```

See Also:

MVIdh_ReadRemoteDataFile

MVIdh_ReadRemoteDataFile

Syntax:

```
int    MVIdh_ReadRemoteDataFile(
                MVIHANDLE handle,
                BYTE node,
                BYTE numElements,
                WORD fileNum,
                BYTE fileType,
                WORD Element,
                BYTE *dataBuf,
                WORD timeout );
```

Parameters:

handle	Handle returned by previous call to MVIdh_Open
node	Node number of remote device to access
numElements	Number of data elements to read from the remote device's data file
fileNum	Number of remote device's data file to access
fileType	Type of remote device's data file to access
Element	Element number of remote device's data file to start reading from
dataBuf	Pointer to buffer into which data is copied from the remote device's data file
timeout	Time to wait for remote device to respond in 100's of milliseconds

MVIdh_ReadRemoteDataFile

Description:

This function copies data into *dataBuf* from a remote device at *node* address. *numElements* data elements will be copied to the application buffer from the remote node's data file. If a response is not received in *timeout*, the function aborts and returns a timeout error. *handle* must be a valid handle returned from MVIdh_Open.

fileNum is the data file number to be read from on the remote device.

fileType is the type of file being accessed. Valid types are listed in table 1.D.

Element is the offset into the data file to start reading data. The number of bytes per element is dependent on the file type and is listed in table 1.D.

Table 1.D - Valid File Types

File Type	Number of bytes per element
MVI_FILETYPE_STATUS	2
MVI_FILETYPE_BIT	2
MVI_FILETYPE_TIMER	6
MVI_FILETYPE_COUNTER	6
MVI_FILETYPE_CONTROL	6
MVI_FILETYPE_INTEGER	2

MVIdh_ReadRemoteDataFile

Return Value:

MVI_SUCCESS	No errors were encountered
MVI_ERR_NOACCESS	*handle* does not have access
MVI_ERR_BADPARAM	Invalid *node*, *offset*, *dataSize*, or *timeout*
MVI_ERR_NOT_MASTER	This function can only be executed in master mode
MVI_ERR_TXCMD_BUSY	Transmitter is already executing a command
MVI_ERR_RM_MSGTIMEOUT	Remote device did not respond in timeout period
MVI_ERR_MEM_ALLOC	Unable to allocate memory for request
MVI_ERR_ILL_CMD_FMT	Target node, illegal command or format
MVI_ERR_ADDRESS_PROBLEM	Target node out of memory, file or rung doesn't exist
MVI_ERR_CMD_EXECUTION	Target node command can't be executed
MVI_ERR_FILE_OPEN	Target node file open by another node
MVI_ERR_PROGRAM_OWNED	Target node program owned by another node

Example:

```
MVIHANDLE   Handle;
BYTE        dataBuf[20];

// Read Remote Data file into buffer

if ( MVI_SUCCESS != MVIdh_ReadRemoteDataFile(Handle, 15, 10, 9,
    MVI_FILETYPE_INTEGER, 0, &databuf[0], 30) )
{
       printf( Read of remote data file failed \n");
}
```

See Also:

MVIdh_WriteRemoteDataFile

Miscellaneous Functions

MVIdh_GetVersionInfo

Syntax:

```
int     MVIdh_GetVersionInfo(MVIHANDLE handle,
                DH485VERSIONINFO *verinfo);
```

Parameters:

handle	Handle returned by previous call to MVIdh_Open
verinfo	Pointer to structure of type DH485VERSIONINFO

Description:

MVIdh_GetVersionInfo retrieves the current version of the DH-485 API library. The information is returned in the structure verinfo. *handle* must be a valid handle returned from MVIdh_Open.

The DH485VERSIONINFO structure is defined as follows:

```
typedef struct tagDH485VERSIONINFO
{
    WORD  APISeries;     /* API series */
    WORD  APIRevision;   /* API revision */
}  DH485VERSIONINFO;
```

Return Value:

MVI_SUCCESS	The version information was read successfully.
MVI_ERR_NOACCESS	*handle* does not have access

Example:

```
MVIHANDLE           Handle;
DH485VERSIONINFO    verinfo;

/* print version of API library */

MVIdh_GetVersionInfo(Handle,&verinfo);
printf("Library Series %d, Rev %d\n", verinfo.APISeries, verinfo.APIRevision);
```

MVIdh_ErrorString

Syntax:

int MVIdh_ErrorString(int errcode, char *buf);

Parameters:

errcode	Error code returned from an API function
buf	Pointer to user buffer to receive message

Description:

MVIdh_ErrorString returns a text error message associated with the error code *errcode*. The null-terminated error message is copied into the buffer specified by *buf*. The buffer should be at least 80 characters in length.

Return Value:

MVI_SUCCESS	Message returned in *buf*
MVI_ERR_BADPARAM	Unknown error code

Example:

```
char   buf[80];
int    rc;

/* print error message */
MVIdh_ErrorString(rc, buf);
printf("Error: %s", buf);
```

Allen-Bradley
Publication Problem Report

If you find a problem with our documentation, please complete and return this form.

Pub. Name ControlLogix Multi-Vendor Interface Module DH-485 API User Manual

Cat. No. 1756-MVI Pub. No. 1756-UM011A-EN-P Pub. Date August 2000 Part No. 957345-91

Check Problem(s) Type:	Describe Problem(s)	Internal Use Only
☐ Technical Accuracy	☐ text ☐ illustration	
☐ Completeness What information is missing?	☐ procedure/step ☐ illustration ☐ definition ☐ example ☐ guideline ☐ feature ☐ explanation ☐ other	☐ info in manual (accessibility) ☐ info not in
☐ Clarity What is unclear?		
☐ Sequence What is not in the right order?		
☐ Other Comments Use back for more comments.		

Your Name _____ Location/Phone _____

Return to: Marketing Communications, Allen-Bradley., 1 Allen-Bradley Drive, Mayfield Hts., OH 44124-6118Phone: (440) 646-3176
FAX: (440) 646-4320

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE

Rockwell Automation

Allen-Bradley

1 ALLEN BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

Reach us now at www.rockwellautomation.com

Wherever you need us, Rockwell Automation brings together leading brands in industrial automation including Allen-Bradley controls, Reliance Electric power transmission products, Dodge mechanical power transmission components, and Rockwell Software. Rockwell Automation's unique, flexible approach to helping customers achieve a competitive advantage is supported by thousands of authorized partners, distributors and system integrators around the world.

Americas Headquarters, 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
European Headquarters SA/NV, avenue Herrmann Debroux, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-UM011A-EN-P - August 2000

PN 957345-91