Allen-Bradley

ControlLogix Multi-Vendor Interface Module DF1 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 DF1 API (Application Programming Interface).

This user manual describes the available software DF1 API libraries and tools, programming information, and example code.

Audience

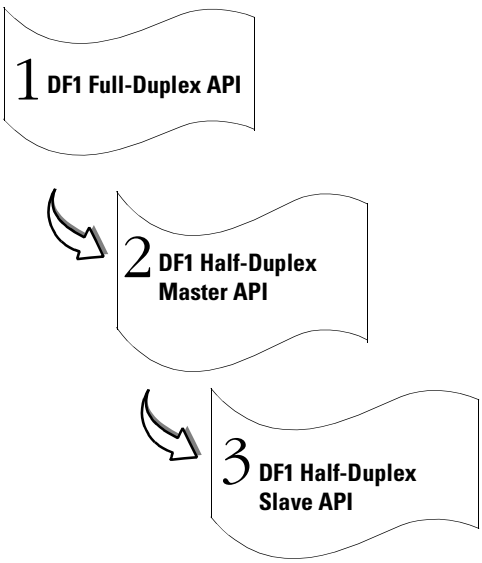
This user manual is intended for control engineers and technicians who are installing, programming, and maintaining a control system that includes a 1756-MVI module.

We assume that you:

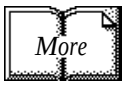
- are familiar with software development in the 16-bit DOS environment using the C programming language.
- are familiar with Allen-Bradley programmable controllers and the ControlLogix platform.

Contents

This user manual contains the following chapters:



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
- DF1 Protocol and Command Set, publication number 1770-6.5.16

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

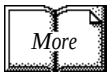
If you find a problem with this manual, please notify us of it on the enclosed Publication Problem Report (at the back of this manual).

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

DF1 Full-Duplex API

The DF1 Full-Duplex (FD) API is one component of the 1756-MVI API Suite. The DF1 FD API allows applications to communicate, via the serial ports, with devices that use the Full-Duplex DF1 protocol. The DF1 FD API functions implement the DF1 FD protocol to the Data-link Layer. Please refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set.



The DF1 FD API provides a common applications interface for all of the Rockwell Automation and third party 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.

DF1 Full-Duplex 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.A.

Table 1.A DF1 Full-Duplex API Functions

Function Category	Function Name	Description
Initialization	MVIdf1_FDOpenPort	Initialize access to a DF1 serial port.
	MVIdf1_FDClosePort	Terminate access to a DF1 serial port.
Communications	MVIdf1_FDGetPkt	Fetch a received packet and/or received packet status
	MVIdf1_FDPutPkt	Place a packet for transmission
	MVIdf1_FDGetPktStat	Fetch the transmission status of a command packet
	MVIdf1_FDGetDiagnostics	Fetch the value of one of the diagnostic counters
Miscellaneous	MVIdf1_FDGetVersionInfo	Get the DF1 FD API version information

Initialization Functions

MVIdf1_FDOpenPort

Syntax:

int MVIdf1_FDOpenPort (FDCFG *DF1Config);

Parameters:

DF1Config Pointer to a structure of type FDCFG. The FDCFG structure is defined below in the Description section.

Description:

MVIdf1_FDOpenPort acquires access to a communications port, configures a DF1 Full-Duplex communications program for that port and then begins execution of that communications program as a background task. This function must be called before any of the other API functions can be used.

The FDCFG structure is defined below

```
typedef struct tagFDCFG
{
BYTE Baud;              // Desired baud rate
BYTE Parity;            // Desired parity
BYTE Stop;              // Desired stop bits
BYTE DupPacket;         // Duplicate packet detection enable/disable
BYTE ErrorDet;          // CRC or BCC error checking selection
BYTE Station;           // Station address. Range: 0-254; 255 = broadcast
BYTE Max_NAKS;          // Number of times a single message will be
                        // transmitted in response to the reception of a
                        // NAK, before being marked as undeliverable.
BYTE Max_ENQS;          // Number of times a single message will be
                        // transmitted in response to a timeout, before
                        // being marked as undeliverable.
BYTE Handshake;         // Hardware handshake control:
                        //      HSHAKE_NONE = none
                        //      HSHAKE_NCC = half-dup, w/o continuous
                        //      carrier
                        //      HSHAKE_CC = half-dup, with continuous
                        //      carrier
WORD RTSSend;           // The RTS send delay in increments of 1 mS
WORD RTSOff;            // The RTS off delay in increments of 1 mS
WORD ACKTimeout;        // The amount of time the unit will wait for an
                        // acknowledgment (DLE-ACK, DLE-NAK to a
                        // transmitted message.
                        // Units are in increments of 1 mS.
BYTE MsgApplTimeout;    // The amount of time the unit will wait for a
                        // reply message in response to an enquiry.
                        // Units are in increments of 1 second.
int  ComPort;           // Set to COM1 or COM2 or COM3
} FDCFG;
```

MVIdf1_FDOpenPort

Baud is the desired baud rate. The allowable values for *Baud* are shown in table 1.B.

Table 1.B - Valid Baud Rates

Baud Rate	Value
BAUD_110	0
BAUD_150	1
BAUD_300	2
BAUD_600	3
BAUD_1200	4
BAUD_2400	5
BAUD_4800	6
BAUD_9600	7
BAUD_19200	8
BAUD_28800	9
BAUD_38400	10
BAUD_57600	11
BAUD_115200	12

Valid values for *Parity* are PARITY_NONE, PARITY_ODD, PARITY_EVEN, PARITY_MARK, and PARITY_SPACE.

The number of stop bits is set by *Stop*. Valid values for *Stop* are STOPBITS1 and STOPBITS2.

DupPacket determines if duplicate packet detection is enabled or disabled. Valid values for *DupPacket* are DUP_PACKET_ENA and DUP_PACKET_DIS. When enabled, a counter indicating the number of duplicate packets received is maintained. See MVIdf1_FDGetDiagnostics.

ErrorDet determines the type of error detection. Valid values for *ErrorDet* are CRC_ERROR_CHK (cyclic redundancy check) and BCC_ERROR_CHK (block check character). A counter indicating the number of packets received with an invalid error check value is maintained. See MVIdf1_FDGetDiagnostics.

Station sets the station number. The valid range is 0-254. Station number 255 should not be used as it is reserved as a broadcast message designation.

Max_NAKS sets the number of NAKs allowed per command. The valid range is 1-255. Once this limit is reached, the status of the message is set to “failed”. See MVIdf1_FDGetPktStat.

Max_ENQS sets the number of ENQs that will be transmitted when an ACK timeout occurs. The range is 1-255. Once this limit is reached, the status of the message is set to “failed”. See MVIdf1_FDGetPktStat.

MVIdf1_FDOpenPort

Handshake determines whether or not hardware handshaking is enabled and if enabled, the type of handshaking used. Valid values are:

HSHAKE_NONE = none
HSHAKE_NCC = half-dup,w/o continuous carrier

RTSSend is the time delay of the transmission of a packet after the RTS output line is activated. This delay only applies when hardware handshaking is active. Units are in increments of 1 mS with a range of 0 - 65535.

RTSOff is the time delay of the deactivation of the RTS output line once a packet has completed transmission. This delay only applies when hardware handshaking is active. Units are in increments of 1 mS with a range of 0 - 65535.

ACKTimeout is the amount of time the unit will wait for an acknowledgment to a transmitted message or an enquiry. The appropriate acknowledgment to a command message is the *DLE-ACK* or *DLE-NAK* sequence. Units are in increments of 1 mS with a range of 0 - 65535.

MsgApplTimeout is the amount of time the unit will wait for a reply message in response to a command. Once the unit successfully issues a command to the peer, the peer must respond with a reply message. This timeout allows sufficient time for the peer to interpret the command, produce a reply, and then transmit that reply. Units are in increments of 1 second with a range of 0 - 255.

ComPort specifies which port is to be opened. The valid values for the 1756-MVI module are COM1 (corresponds to PRT1), COM2 (corresponds to PRT2), and COM3 (corresponds to PRT3).

Note: If the console is enabled or the Setup jumper is installed, the baud rate for COM1 is set as configured in BIOS Setup and cannot be changed by MVIdf1_FDOpenPort. MVIdf1_FDClosePort will return *MVI_SUCCESS*, but the baud rate will not be affected. The console should be disabled in BIOS Setup if COM1 is to be accessed with the DF1 FD API.

IMPORTANT Once the DF1 port has been opened, MVIdf1_FDClosePort must always be called before exiting the application.

MVIdf1_FDOpenPort

Return Value:

MVI_SUCCESS	DF1 port was opened successfully
MVI_ERR_REOPEN	DF1 port is already open
MVI_ERR_NODEVICE	UART not found on port
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_OS	Error occurred in DOS 6-XL operating system

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

Example:

```
FDSCFG Port1Cfg;

if (MVIdf1_FDOpenPort(&Port1Cfg) != MVI_SUCCESS) {
    printf("Open failed!\n");
} else {
    printf("Open succeeded\n");
}
```

See Also:

MVIdf1_FDClosePort

MVIdf1_FDClosePort

Syntax:

int MVIdf1_FDClosePort (BYTE comport);

Parameters:

comport DF1 port to close

Description:

MVIdf1_FDClosePort is used by the application to release control of the designated communications port. The application must have previously opened the comport with the MVIdf1_FDOpenPort API.

IMPORTANT Once the DF1 port has been opened, this function must always be called before exiting the application.

Return Value:

MVI_SUCCESS	DF1 port was closed successfully
MVI_ERR_NOACCESS	DF1 has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
MVIdf1_FDClosePort(COM1);
```

See Also:

MVIdf1_FDOpenPort

Communications

MVIdf1_FDGetPkt

Syntax:

int MVIdf1_FDGetPkt (BYTE comport, BYTE *DF1_Pkt, WORD *length, RSPRCV *DF1_Stat);

Parameters:

comport	DF1 port from which to fetch a packet/status.
DF1_Pkt	pointer to the array into which the packet is to be stored.
length	pointer to the variable into which the length of the packet is to be stored.
DF1_Stat	pointer to a structure of type RSPRCV. This is the pointer to the array into which the response packet status will be stored.

Description:

MVIdf1_FDGetPkt determines if a response packet/status is available from the designated port. The received packet may be a response to a command or it may be a peer-initiated packet.

The status of the response packet, returned in *DF1_Stat*, is critical to the application for proper processing of the response packet:

- If the *RespStatus* field of *DF1_Stat* is equal to RESP_VALID, then the application layer packet in *DF1_Pkt* is valid and the length of the packet is contained in *length*. Also, the *Src*, *Cmd,* and *TNS* fields of *DF1_Stat* contain the source, command, and transaction number for the received packet.

- If the *RespStatus* field of *DF1_Stat* is equal to RESP_TIMEOUT, the application layer packet in *DF1_Pkt*, as well as the *length* field, is invalid. In this case, the *Src*, *Cmd*, and *TNS* fields of *DF1_Stat* contain the source, command, and transaction number of a command which has not received a reply packet within the time limit determined by the parameter *MsgApplTimeout*. See MVIdf1_FDOpenPort.

Each call to MVIdf1_FDGetPkt returns a single response packet/status. To retrieve all current response packet/status information the application should continue to call MVIdf1_FDGetPkt until the return value is MVI_ERR_NODATA. Once the response packet/status has been retrieved, it is longer available to the application; therefore, the application must process each packet/status immediately after it is retrieved.

MVIdf1_FDGetPkt

DF1_Pkt is a pointer to an array where the application layer data will be stored. Only the application layer data will be stored to this array, not the entire DF1 packet.

length is a pointer where the length of the returned packet will be stored.

DF1_Stat is a pointer to a structure of type RSPRCV. The RSPRCV structure is defined below:

```
typedef struct tagRSPRCV
{
    BYTE Src;          // Source address of received response packet
    BYTE Cmd;          // Command value of received response packet
    WORD TNS;          // TNS count for response packet
    BYTE RespStatus;   // Receive status of the response packet
}RSPRCV;
```

Src The source node value of the reply packet. Note: this is the destination node value in the associated command packet.

Cmd The command code of the reply packet.

TNS The transaction number of the reply packet

RespStatus returned will be one of the following:

- RESP_VALID = The data-link layer packet in *DF1_Pkt* is a valid packet and the length of the packet is contained in **length*. Also, the *Src*, *Cmd*, and *TNS* fields of *DF1_Stat* contain the source, command and transaction number for the response packet.

- RESP_TIMEOUT = The data-link layer packet in *DF1_Pkt* and the length are invalid. The *Src, Cmd,* and *TNS* fields of *DF1_Stat* are associated with a command packet which has not received a reply packet within the time limit determined by the parameter *MsgApplTimeout*. See MVIdf1_FDOpenPort.

Return Value:

MVI_SUCCESS	Packet/status retrieved successfully
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_NODATA	No packet/status available

MVIdf1_FDGetPkt

Example:

```
BYTE        DF1_Pkt[MAX_DF1_BUFR];
RSPRCV      DF1_Stat;
WORD        length;

if (MVIdf1_FDGetPkt(COM1,DF1_Pkt,&length,&DF1_Stat) ==
        MVI_SUCCESS) {
    printf ("Received packet/status available. \n");
}
```

See Also:

MVIdf1_FDPutPkt

MVIdf1_FDPutPkt

Syntax:

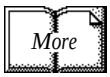
int MVIdf1_FDPutPkt (BYTE comport, BYTE *DF1_Pkt, WORD *length);

Parameters:

comport	DF1 port to which to send a packet.
DF1_Pkt	pointer to array from which the packet is to be retrieved.
length	pointer to variable from which the length of the packet will be retrieved.

Description:

MVIdf1_FDPutPkt takes the application layer data from the array pointed to by *DF1_Pkt* and places it into the source buffer for transmission. The length of the data is the variable pointed to by *length*. The data passed to this function is only the application layer data, not the entire DF1 packet.



Please refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set.

DF1_Pkt is a pointer to an array from which the application layer data will be retrieved. The application should store only the application layer data, not the entire DF1 packet.

length is a pointer to the variable that contains the length of the packet to be stored for transmission.

Return Value:

MVI_SUCCESS	Packet stored successfully for transmission
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
BYTE        DF1_Pkt[MAX_DF1_BUFR];
WORD        length;

if (MVIdf1_FDPutPkt(COM1, DF1_Pkt,&length) == MVI_SUCCESS) {
   printf ("Packet stored for transmission. \n");

}
```

See Also:

MVIdf1_FDGetPkt

MVIdf1_FDGetPktStat

MVIdf1_FDGetPktStat

Syntax:

int MVIdf1_FDGetPktStat (BYTE comport, SRCXMT *DF1_Stat);

Parameters:

comport	DF1 port on which to request packet status.
DF1_Stat	pointer to a structure of type SRCXMT. The SRCXMT structure is defined below in the Description.

Description:

This function returns the status of a packet that has been placed into the source buffer, by the MVIdf1_FDPutPkt function, for transmission. A transmit status queue is maintained to provide the application with information for each packet placed in the source buffer for transmission.

The transmit status queue contains the status of each packet which has terminated transmission (pass or fail) as well as the status of a packet which may be in the process of being transmitted. The status of packets that have not yet begun transmission will not be in the queue.

Each call to this function returns the status of one packet. Once the status of a packet is reported, that packet's status is removed from the queue and the next query will return the status of the next packet in the status queue.

The last packet status in the queue may be the status of a packet in the process of being transmitted. This packet's status will be returned but it will not be removed from the queue until the query is made when the packet's transmission has been terminated.

The SRCXMT structure is defined below:

```
typedef struct tagSRCXMT
{
        BYTE Src;          // Source node of application layer data
        BYTE Cmd;          // Command code of application layer data
        WORD TNS;          // TNS number for application packet
        BYTE XmitStatus;   // Transmit status of the packet
}SRCXMT;
```

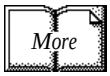
Src	The source node value found in the application layer data of the packet.
Cmd	The command code found in the application layer data of the packet.
TNS	The transaction number found in the application layer data of the packet.

MVIdf1_FDGetPktStat

XmitStatus The status of the packet, where:

MVIDF1_XMITTING	= packet currently being transmitted
MVIDF1_SUCCESS	= packet has been successfully transmitted
MVIDF1_FAILED	= packet transmission failed

Note: The *Src*, *Cmd*, and *TNS* values are to be used by the application to identify the packet to which the status information, *XmitStatus*, pertains.



Please refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set, for a detailed explanation of the *Src*, *Cmd*, and *TNS* elements of the application layer data.

Return Value:

MVI_SUCCESS	Packet stored successfully for transmission
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_NOSTAT	No packet status available

Example:

```
SRCXMTDF1_Stat;

if (MVIdf1_FDGetPktStat(COM1,&DF1_Stat) == MVI_SUCCESS) {
    printf ("Packet transmission status available. \n");
}
```

See Also:

MVIdf1_FDPutPkt

MVIdf1_FDGetDiagnostics

Syntax:

int MVIdf1_FDGetDiagnostics (BYTE comport, WORD *DF1_Diag, BYTE DF1_DiagNum,BYTE reset);

Parameters:

comport	DF1 port from which to fetch a diagnostic counter value
DF1_Diag	pointer to the variable in which to store the counter value
DF1_DiagNum	number of diagnostic counter to retrieve
reset	reset/no-reset flag for diagnostic counter

Description:

MVIdf1_FDGetDiagnostics retrieves the value of the designated diagnostic counter. Depending on the value of *reset*, the counter may or may not be reset to a value of zero.

The diagnostic counters may be used by the application to track and analyze communications problems, monitor packet flow, and to allow link optimization.

DF1_Diag is a pointer to a variable in which to store the diagnostic counter value. These counters will roll over to a value of zero if allowed to increment without monitoring and control.

DF1_DiagNum is the number of the desired diagnostic counter value, where:

PACKETS_RCVD = 0x00 = Number of valid packets received from the peer units.

BAD_CRC_BCC = 0x03 = Number of packets received with invalid error checks.

DUPS_RCVD = 0x04 = Number of duplicate packets received. This counter is only active if duplicate packet detection is enabled.

PACKETS_XMITTED = 0x05 = Total number of packets transmitted. This value includes message re-transmissions.

SINK_FULL = 0x06 = Number of received packets which have been rejected because the sink (receive) buffer is full. This number indicates that the application may not be retrieving received packets in a timely manner.

SOURCE_FULL = 0x07 = Number of transmit packets which have been rejected because the source (transmit) buffer is full. This number may indicate a problem with modem handshaking.

MESSAGE_RETRIES = 0x08 = Number of packets which have been re-transmitted.

reset determines if the diagnostic counter is or is not reset to a value of zero. A value of non-zero will reset the counter.

Return Value:

MVI_SUCCESS	Packet retrieved successfully
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
WORDDF1_Diag;

if (MVIdf1_FDGetDiagnostics(COM1,&DF1_Diag,PACKETS_RCVD,0x00) ==
MVI_SUCCESS) {
   printf ("Total number of packets received is %u,\n", DF1_Diag);
// The counter has not been reset
}
```

MVIdf1_Get VersionInfo

Syntax:

int MVIdf1_FDGetVersionInfo(MVIDF1VERSIONINFO *verinfo);

Parameters:

verinfo pointer to structure of type MVIDF1VERSIONINFO

Description:

MVIdf1_FDGetVersionInfo retrieves the current version of the API. The version information is returned in the structure *verinfo*.

The MVIDF1VERSIONINFO structure is defined as follows:

```
typedef struct tagMVIDF1VERSIONINFO
{
WORDAPISeries;/* API series */
WORDAPIRevision;/* API revision */
} MVIDF1VERSIONINFO;
```

Return Value:

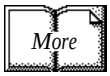
MVI_SUCCESS The version information was read successfully.

Example:

```
MVIDF1VERSIONINFO  verinfo;

/* print version of API library */
MVIdf1_FDGetVersionInfo(&verinfo);
printf("Library Series %d, Rev %d\n", verinfo.APISeries, verinfo.APIRevision);
```

DF1 Half-Duplex Master API



The DF1 Half-Duplex Master (HDM) API is one component of the 1756-MVI API Suite. The DF1 HDM API allows applications to communicate, via the serial ports, with devices that use the Half-Duplex Slave DF1 protocol. The DF1 HDM API functions implement the DF1 HDM protocol to the Data-link Layer. Please refer to the Allen-Bradley publication number 1770-6.5.16, titled DF1 Protocol and Command Set.

The DF1 HDM API provides a common applications interface for all of the Rockwell Automation and third party 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.

DF1 Half-Duplex Master 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 2.A.

Table 2.A DF1 Half-Duplex Master API Functions

Function Category	Function Name	Description
Initialization	MVIdf1_HDMOpenPort	Initialize access to a DF1 serial port.
	MVIdf1_HDMClosePort	Terminate access to a DF1 serial port.
Communications	MVIdf1_HDMGetRespPkt	Fetch a response packet and/or response packet status
	MVIdf1_HDMPutPkt	Place a packet for transmission
	MVIdf1_HDMGetPktStat	Fetch the status of a transmitted packet
	MVIdf1_HDMGetDiagnostics	Fetch the value of one of the diagnostic counters
Miscellaneous	MVIdf1_HDMGetVersionInfo	Get the DF1 HDM API version information

Initialization Functions

MVIdf1_HDMOpenPort

Syntax:

int MVIdf1_HDMOpenPort (HDMCFG *DF1Config);

Parameters:

DF1Config Pointer to a structure of type HDMCFG. The HDMCFG structure is defined below in the Description section.

Description:

MVIdf1_HDMOpenPort acquires access to a communications port, configures a DF1 Half-Duplex Master communications program for that port and then begins execution of that communications program as a background task. This function must be called before any of the other API functions can be used.

MVIdf1_HDMOpenPort

The HDMCFG structure is defined below

```
typedef struct tagHDMCFG
{
BYTE Baud;                  // Desired baud rate
BYTE Parity;                // Desired parity
BYTE Stop;                  // Desired stop bits
BYTE DupPacket;             // Duplicate packet detection enable/disable
BYTE ErrorDet;              // CRC or BCC error checking selection
BYTE Station;               // Station address. Range: 0-254; 255 = broadcast
BYTE MsgRetries;            // Number of times a single message will be
                            // transmitted to a slave before being marked as
                            // undeliverable.
BYTE Handshake;             // Hardware handshake control:
                            //      HSHAKE_NONE =none
                            //      HSHAKE_NCC =half-dup, w/o continuous
                            //      carrier
                            //      HSHAKE_CC =half-dup, with continuous
                            //      carrier
WORD RTSSend;               // The RTS send delay in increments of 1 mS
WORD RTSOff;                // The RTS off delay in increments of 1 mS
WORD ReplyMsgWait;          // The amount of time the master will wait
                            // after receiving an ACK (to a master-initiated
                            // message) before polling the slave for a reply.
                            // Units are in increments of 1 mS. This is
                            // applicable only for the message-based polling
                            // modes
WORD ACKTimeout;            // The amount of time the master will wait for an
                            // acknowledgment (DLE-ACK, DLE-EOT or a
                            // packet) to a transmitted message or an
                            // enquiry. Units are in increments of 1 mS.
BYTE MsgApplTimeout;        // The amount of time the master will wait for a
                            // reply message in response to an enquiry.
                            // Units are in increments of 1 second.
BYTE PollingMode;           // The polling method to use:
                            //      MSG_NO_SLAVE_ALLD = Message based,
                            //      no slave initiated messages allowed.
                            //      MSG_SLAVE_ALLD = Message based,
                            //      slave initiated messages are allowed.
                            //      STD_SINGLE = Standard, single response
                            //      message during a polling cycle.
                            //      STD_MULTIPLE = Standard, multiple
                            //      responses during a polling cycle.
BYTE Norm_Poll_Low_Addr;    // The Normal Polling Range Low Address.
BYTE Norm_Poll_High_Addr;   // The Normal Polling Range High Address.
BYTE Norm_Poll_Group_Size;  // The number of active nodes in the normal
                            // poll list to be polled during a single pass of
                            // the normal polling cycle.
BYTE Priority_Poll_Low_Addr;// The Priority Polling Range Low Address.
BYTE Priority_Poll_High_Addr;// The Priority Polling Range High Address.
int  ComPort;               // Set to COM1 or COM2 or COM3
} HDMCFG;
```

MVIdf1_HDMOpenPort

Baud is the desired baud rate. The allowable values for *Baud* are shown in table 2.B.

Table 2.B - Valid Baud Rates

Baud Rate	Value
BAUD_110	0
BAUD_150	1
BAUD_300	2
BAUD_600	3
BAUD_1200	4
BAUD_2400	5
BAUD_4800	6
BAUD_9600	7
BAUD_19200	8
BAUD_28800	9
BAUD_38400	10
BAUD_57600	11
BAUD_115200	12

Valid values for *Parity* are PARITY_NONE, PARITY_ODD, PARITY_EVEN, PARITY_MARK, and PARITY_SPACE.

The number of stop bits is set by *Stop*. Valid values for *Stop* are STOPBITS1 and STOPBITS2.

DupPacket determines if duplicate packet detection is enabled or disabled. Valid values for *DupPacket* are DUP_PACKET_ENA and DUP_PACKET_DIS. When enabled, a counter indicating the number of duplicate packets received is maintained. See MVIdf1_HDSGetDiagnostics.

ErrorDet determines the type of error detection. Valid values for *ErrorDet* are CRC_ERROR_CHK (cyclic redundancy check) and BCC_ERROR_CHK (block check character). A counter indicating the number of packets received with an invalid error check value is maintained. See MVIdf1_HDSGetDiagnostics.

Station sets the station number. The valid range is 0-254. Station number 255 should not be used as it is reserved as a broadcast message designation.

MsgRetries sets the number of times a single message will be transmitted in response to a poll from the master. The range is 1-255. Once this limit is reached, the status of the message is set to “failed”. See MVIdf1_HDSGetPktStat.

MVIdf1_HDMOpenPort

Handshake determines whether or not hardware handshaking is enabled and if enabled, the type of handshaking used. Valid values are:

HSHAKE_NONE = none
HSHAKE_NCC = half-dup,w/o continuous carrier
HSHAKE_CC = half-dup, with continuous carrier

RTSend is the time delay of the transmission of a packet after the RTS output line is activated. This delay only applies when hardware handshaking is active. Units are in increments of 1 mS with a range of 0 - 65535.

RTSOff is the time delay of the deactivation of the RTS output line once a packet has completed transmission. This delay only applies when hardware handshaking is active. Units are in increments of 1 mS with a range of 0 - 65535.

ReplyMsgWait is the amount of time the master will wait, after receiving an ACK (to a master-initiated message), before polling the slave for a reply. This parameter is only applicable when a message-based polling method is used. Units are in increments of 1 mS with a range of 0 - 65535.

ACKTimeout is the amount of time the master will wait for an acknowledgment to a transmitted message or an enquiry. The appropriate acknowledgment to a command message is the DLE-ACK sequence. The appropriate acknowledgment to an enquiry (DLE-ENQ sequence) is either a response packet or the DLE-EOT sequence. Units are in increments of 1 mS with a range of 0 - 65535.

MsgApplTimeout is the amount of time the master will wait for a reply message in response to a command from the master. Once the master successfully issues a command to a slave, the slave must respond with a reply message. This timeout allows sufficient time for the slave to interpret the command, produce a reply and then transmit that reply when the master queries the slave. Units are in increments of 1 second with a range of 0 - 255.

PollingMode is the polling method to use:

STD_SINGL = Standard, single response message during a polling cycle.

STD_MULTIPLE = Standard, multiple responses during a polling cycle.

MSG_NO_SLAVE_ALLD = Message based, no slave initiated messages allowed.

MSG_SLAVE_ALLD = Message based, slave initiated messages are allowed.

MVIdf1_HDMOpenPort

Norm_Poll_Low_Addr is the numerically lowest slave station number at which to begin the normal polling cycle. This value, along with the Norm_Poll_High_Addr parameter, determines the address range of the slave stations which will be included in the normal polling cycle.

Norm_Poll_High_Addr is the numerically highest slave station number at which to terminate the normal polling cycle. This value, along with the Norm_Poll_Low_Addr parameter, determines the address range of the slave stations which will be included in the normal polling cycle.

Norm_Poll_Group_Size is the number of stations to be polled during a single pass of the normal polling cycle.

Priority_Poll_Low_Addr is the numerically lowest slave station number at which to begin the priority polling cycle. This value, along with the Priority_Poll_High_Addr parameter, determines the address range of slave stations which will be included in the priority polling cycle.

Priority_Poll_High_Addr is the numerically highest slave station number at which to terminate the priority polling cycle. This value, along with the Priority_Poll_Low_Addr parameter, determines the address range of slave stations which will be included in the priority polling cycle.

ComPort specifies which port is to be opened. The valid values for the 1756-MVI module are COM1 (corresponds to PRT1), COM2 (corresponds to PRT2), and COM3 (corresponds to PRT3).

Note: If the console is enabled or the Setup jumper is installed, the baud rate for COM1 is set as configured in BIOS Setup and cannot be changed by MVIdf1_HDMOpenPort. MVIdf1_HDMClosePort will return MVI_SUCCESS, but the baud rate will not be affected. The console should be disabled in BIOS Setup if COM1 is to be accessed with the DF1 HDM API.

IMPORTANT Once the DF1 port has been opened, MVIdf1_HDSClosePort should always be called before exiting the application.

Return Value:

MVI_SUCCESS	DF1 port was opened successfully
MVI_ERR_REOPEN	DF1 port is already open
MVI_ERR_NODEVICE	UART not found on port
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_OS	Error occurred in DOS 6-XL operating system

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

MVIdf1_HDMOpenPort

Example:

```
HDSCFG Port1Cfg;

if (MVIdf1_HDMOpenPort(&Port1Cfg) != MVI_SUCCESS) {
    printf("Open failed!\n");
} else {
    printf("Open succeeded\n");
}
```

See Also:

MVIdf1_HDMClosePort

MVIdf1_HDMClosePort

Syntax:

int MVIdf1_HDMClosePort (BYTE comport);

Parameters:

comport DF1 port to close

Description:

MVIdf1_HDMClosePort is used by the application to release control of the designated communications port. The application must have previously opened the comport with the MVIdf1_HDMOpenPort API.

IMPORTANT Once the DF1 port has been opened, this function should always be called before exiting the application.

Return Value:

MVI_SUCCESS	DF1 port was closed successfully
MVI_ERR_NOACCESS	DF1 has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
MVIdf1_HDMClosePort(COM1);
```

See Also:

MVIdf1_HDMOpenPort

Communications

MVIdf1_HDMGetRespPkt

Syntax:

int MVIdf1_HDMGetRespPkt (BYTE comport, BYTE *DF1_Pkt, WORD *length, RSPRCV *DF1_Stat);

Parameters:

comport	DF1 port from which to fetch a packet/status.
DF1_Pkt	pointer to the array into which the packet is to be stored.
length	pointer to the variable into which the length of the packet is to be stored.
DF1_Stat	pointer to a structure of type RSPRCV. This is the pointer to the array into which the response packet status will be stored.

Description:

MVIdf1_HDMGetRespPkt determines if a response packet/status is available from the designated port. A response packet is a packet returned from a slave in response to an enquiry by the master. The response packet may be a reply packet to a command or it may be a slave-initiated packet. Slave-to-slave packets will not be passed to the application.

The status of the response packet, returned in *DF1_Stat*, is critical to the application for proper processing of the response packet:

- If the *RespStatus* field of *DF1_Stat* is equal to RESP_VALID, then the application layer packet in *DF1_Pkt* is valid and the length of the packet is contained in length. Also, the *Src*, *Cmd* and *TNS* fields of *DF1_Stat* contain the source, command and transaction number for the response packet.

- If the *RespStatus* field of *DF1_Stat* is equal to RESP_TIMEOUT, the application layer packet in *DF1_Pkt*, as well as the length field, is invalid. In this case, the *Src*, *Cmd*, and *TNS* fields of *DF1_Stat* contain the source, command, and transaction number of a command which has not received a reply packet within the time limit determined by the parameter *MsgApplTimeout*. See MVIdf1_HDMOpenPort.

MVIdf1_HDMGetRespPkt

Each call to MVIdf1_HDMGetRespPkt returns a single response packet/status. To retrieve all current response packet/status information the application should continue to call MVIdf1_HDMGetRespPkt until the return value is MVI_ERR_NODATA. Once the response packet/status has been retrieved, it is longer available to the application; therefore, the application must process each packet/status immediately after it is retrieved.

DF1_Pkt is a pointer to an array where the application layer data will be stored. Only the application layer data will be stored to this array, not the entire DF1 packet.

length is a pointer where the length of the returned packet will be stored.

DF1_Stat is a pointer to a structure of type RSPRCV. The RSPRCV structure is defined below:

```
typedef struct tagRSPRCV
{
    BYTE Src;        // Source address of received response packet
    BYTE Cmd;        // Command value of received response packet
    WORD TNS;        // TNS count for response packet
    BYTE RespStatus; // Receive status of the response packet
}RSPRCV;
```

Src The source node value of the reply packet. Note: this is the destination node value in the associated command packet.

Cmd The command code of the reply packet.

TNS The transaction number of the reply packet

RespStatus returned will be one of the following:

- RESP_VALID = The data-link layer packet in *DF1_Pkt* is a valid packet and the length of the packet is contained in *length. Also, the *Src*, *Cmd*, and *TNS* fields of *DF1_Stat* contain the source, command and transaction number for the response packet.

- RESP_TIMEOUT = The data-link layer packet in *DF1_Pkt* and the length are invalid. The *Src, Cmd,* and *TNS* fields of *DF1_Stat* are associated with a command packet which has not received a reply packet within the time limit determined by the parameter *MsgApplTimeout*. See MVIdf1_HDMOpenPort.

MVIdf1_HDMGetRespPkt

Return Value:

MVI_SUCCESS	Packet retrieved successfully
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_NODATA	No packet available

Example:

```
BYTE        DF1_Pkt[MAX_DF1_BUFR];
RSPRCV      DF1_Stat;
WORD        length;

if (MVIdf1_HDMGetRespPkt(COM1,DF1_Pkt,&length,&DF1_Stat) ==
        MVI_SUCCESS) {
    printf ("Response packet/status available. \n");
}
```

See Also:

MVIdf1_HDMPutPkt

MVIdf1_HDMPutPkt

Syntax:

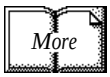
int MVIdf1_HDMPutPkt (BYTE comport, BYTE *DF1_Pkt, WORD *length);

Parameters:

comport	DF1 port to which to send a packet.
DF1_Pkt	pointer to array from which the packet is to be retrieved.
length	pointer to variable from which the length of the packet will be retrieved.

Description:

MVIdf1_HDMPutPkt takes the application layer data from the array pointed to by DF1_Pkt and places it into the source buffer for transmission. The length of the data is the variable pointed to by length. The data passed to this function is only the application layer data, not the entire DF1 packet.



Please refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set.

DF1_Pkt is a pointer to an array from which the application layer data will be retrieved. The application should store only the application layer data, not the entire DF1 packet.

length is a pointer to the variable that contains the length of the packet to be stored for transmission.

Return Value:

MVI_SUCCESS	Packet stored successfully for transmission
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
BYTE    DF1_Pkt[MAX_DF1_BUFR];
WORD    length;

if (MVIdf1_HDMPutPkt(COM1,DF1_Pkt,&length) == MVI_SUCCESS) {
   printf ("Packet stored for transmission. \n");

}
```

See Also:

MVIdf1_HDMGetRespPkt

MVIdf1_HDMGetPktStat

MVIdf1_HDMGetPktStat

Syntax:

int MVIdf1_HDMGetPktStat (BYTE comport, SRCXMT *DF1_Stat);

Parameters:

comport	DF1 port on which to request packet status.
DF1_Stat	pointer to a structure of type SRCXMT. The SRCXMT structure is defined below in the Description.

Description:

This function returns the status of a packet that has been placed into the source buffer, by the MVIdf1_HDMPutPkt function, for transmission. A transmit status queue is maintained to provide the application with information for each packet placed in the source buffer for transmission.

The transmit status queue contains the status of each packet which has terminated transmission (pass or fail) as well as the status of a packet which may be in the process of being transmitted. The status of packets that have not yet begun transmission will not be in the queue.

Each call to this function returns the status of one packet. Once the status of a packet is reported, that packet's status is removed from the queue and the next query will return the status of the next packet in the status queue.

The last packet status in the queue may be the status of a packet in the process of being transmitted. This packet's status will be returned but it will not be removed from the queue until the query is made when the packet's transmission has been terminated.

The SRCXMT structure is defined below:

```
typedef struct tagSRCXMT
{
        BYTE Src;          // Source node of application layer data
        BYTE Cmd;          // Command code of application layer data
        WORD TNS;          // TNS number for application packet
        BYTE XmitStatus;   // Transmit status of the packet
}SRCXMT;
```

Src	The source node value found in the application layer data of the packet.
Cmd	The command code found in the application layer data of the packet.
TNS	The transaction number found in the application layer data of the packet.

MVIdf1_HDMGetPktStat

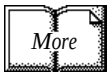
XmitStatus The status of the packet, where:

MVIDF1_XMITTING = packet currently being transmitted.

MVIDF1_SUCCESS = packet has been successfully transmitted.

MVIDF1_FAILED = packet transmission failed.

Note: The *Src*, *Cmd*, and *TNS* values are to be used by the application to identify the packet to which the status information, *XmitStatus*, pertains.



Please refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set, for a detailed explanation of the *Src*, *Cmd*, and *TNS* elements of the application layer data.

Return Value:

MVI_SUCCESS	Packet stored successfully for transmission
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_NOSTAT	No packet status available

Example:

```
SRCXMTDF1_Stat;

if (MVIdf1_HDMPutPkt(COM1,&DF1_Stat) == MVI_SUCCESS) {
    printf ("Packet status available. \n");
}
```

See Also:

MVIdf1_HDMPutPkt

MVIdf1_HDMGetDiagnostics

Syntax:

int MVIdf1_HDMGetDiagnostics (BYTE comport, WORD *DF1_Diag, BYTE DF1_DiagNum,BYTE reset);

Parameters:

comport	DF1 port from which to fetch a diagnostic counter value
DF1_Diag	pointer to the variable in which to store the counter value
DF1_DiagNum	number of diagnostic counter to retrieve
reset	reset/no-reset flag for diagnostic counter

Description:

MVIdf1_HDMGetDiagnostics retrieves the value of the designated diagnostic counter. Depending on the value of reset, the counter may or may not be reset to a value of zero.

The diagnostic counters may be used by the application to track and analyze communications problems, monitor packet flow, and to allow link optimization.

DF1_Diag is a pointer to a variable in which to store the diagnostic counter value. These counters will roll over to a value of zero if allowed to increment without monitoring and control.

DF1_DiagNum is the number of the desired diagnostic counter value where:

PACKETS_RCVD = 0x00 = Number of valid packets received from the DF1 master.

POLLS_RCVD = 0x01 = Number of polls received from the DF1 master.

NAKS_RCVD = 0x02 = Number of NAKs received from the DF1 master.

BAD_CRC_BCC = 0x03 = Number of packets received with invalid error checks.

DUPS_RCVD = 0x04 = Number of duplicate packets received. This counter is only active if duplicate packet detection is enabled.

PACKETS_XMITTED = 0x05 = Total number of packets transmitted. This value includes message re-transmissions.

SINK_FULL = 0x06 = Number of received packets which have been rejected because the sink (receive) buffer is full. This number indicates that the application may not be retrieving received packets in a timely manner.

MVIdf1_HDMGetDiagnostics

SOURCE_FULL = 0x07 = Number of transmit packets which have been rejected because the source (transmit) buffer is full. This number indicates that the DF1 master may not be polling the DF1 slave in a timely manner.

MESSAGE_RETRIES = 0x08 = Number of packets which have been re-transmitted.

reset determines if the diagnostic counter is or is not reset to a value of zero. A value of non-zero will reset the counter.

Return Value:

MVI_SUCCESS	Packet retrieved successfully
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
WORDDF1_Diag;
if (MVIdf1_HDMGetDiagnostics(COM1,&DF1_Diag,PACKETS_RCVD,0x00) ==
MVI_SUCCESS) {
   printf ("Total number of packets received is %u, \n", DF1_Diag);
// The counter has not been reset
}
```

MVIdf1_Get VersionInfo

Syntax:

int MVI df1_GetVersionInfo(MVIDF1VERSIONINFO *verinfo);

Parameters:

verinfo pointer to structure of type MVIDF1VERSIONINFO

Description:

MVI df1_GetVersionInfo retrieves the current version of the API. The version information is returned in the structure *verinfo*.

The MVIDF1VERSIONINFO structure is defined as follows:

```
typedef struct tagMVIDF1VERSIONINFO
{
WORDAPISeries;/* API series */
WORDAPIRevision;/* API revision */
} MVIDF1VERSIONINFO;
```

Return Value:

MVI_SUCCESS The version information was read successfully.

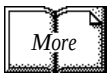
Example:

```
MVIDF1VERSIONINFO  verinfo;

/* print version of API library */
MVI df1_GetVersionInfo(&verinfo);
printf("Library Series %d, Rev %d\n", verinfo.APISeries, verinfo.APIRevision);
```

DF1 Half-Duplex Slave API

The DF1 Half-Duplex Slave (HDS) API is one component of the 1756-MVI API Suite. The DF1 HDS API allows applications to communicate, via the serial ports, with a device that uses the Half-Duplex Master DF1 protocol. The DF1 HDS API functions implement the DF1 HDS protocol to the Data-link Layer.



Please refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set.

The DF1 HDS API provides a common applications interface for all of the Rockwell Automation and third party 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.

DF1 Half-Duplex Slave 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 3.A.

Table 3.A DF1 Half-Duplex Slave API Functions

Function Category	Function Name	Description
Initialization	MVIdf1_HDSOpenPort	Initialize access to a DF1 serial port.
	MVIdf1_HDSClosePort	Terminate access to a DF1 serial port.
Communications	MVIdf1_HDSGetPkt	Fetch a received packet
	MVIdf1_HDSPutPkt	Place a packet for transmission
	MVIdf1_HDSGetPktStat	Fetch the status of a transmitted packet
	MVIdf1_HDSGetDiagnostics	Fetch the value of one of the diagnostic counters
Miscellaneous	MVIdf1_HDSGetVersionInfo	Get the DF1 HDS API version information

Initialization Functions

MVIdf1_HDSOpenPort

Syntax:

int MVIdf1_HDSOpenPort (HDSCFG *DF1Config);

Parameters:

DF1Config Pointer to a structure of type HDSCFG. The HDSCFG structure is defined below in the Description section.

Description:

MVIdf1_HDSOpenPort acquires access to a communications port, configures a DF1 Half-Duplex Slave communications program for that port and then begins execution of that communications program as a background task. This function must be called before any of the other API functions can be used.

The HDSCFG structure is defined below:

```
typedef struct tagHDSCFG
{
        BYTE Baud;          // Desired baud rate
        BYTE Parity;        // Desired parity
        BYTE Stop;          // Desired stop bits
        BYTE DupPacket;     // Duplication packet detection enable/disable
        BYTE ErrorDet;      // Error detection selection: BCC or CRC
        BYTE Station;       // Station address.
        BYTE MsgRetries;    // Number of times a single message will be
                            // transmitted in response to a poll from the master
        BYTE Handshake;     // Hardware handshake control;
                            //      0=none
                            //      1=half-dup, w/o continuous carrier
                            //      2=half-dup, with continuous carrier
        WORD RTSOn;         // The RTS send delay in increments of X mS
        WORD RTSOff;        // The RTS off delay in increments of X mS
        int  ComPort;       //
} HDSCFG;
```

ComPort specifies which port is to be opened. The valid values for the 1756AV-MVI module are COM1 (corresponds to PRT1), COM2 (corresponds to PRT2), and COM3 (corresponds to PRT3).

MVIdf1_HDSOpenPort

Baud is the desired baud rate. The allowable values for *Baud* are shown in table 3.B.

Table 3.B - Valid Baud Rates

Baud Rate	Value
BAUD_110	0
BAUD_150	1
BAUD_300	2
BAUD_600	3
BAUD_1200	4
BAUD_2400	5
BAUD_4800	6
BAUD_9600	7
BAUD_19200	8
BAUD_28800	9
BAUD_38400	10
BAUD_57600	11
BAUD_115200	12

Valid values for *Parity* are PARITY_NONE, PARITY_ODD, PARITY_EVEN, PARITY_MARK, and PARITY_SPACE.

The number of stop bits is set by *Stop*. Valid values for *Stop* are STOPBITS1 and STOPBITS2.

DupPacket determines if duplicate packet detection is enabled or disabled. Valid values for *DupPacket* are DUP_PACKET_ENA and DUP_PACKET_DIS. When enabled, a counter indicating the number of duplicate packets received is maintained. See MVIdf1_HDSGetDiagnostics.

ErrorDet determines the type of error detection. Valid values for *ErrorDet* are CRC_ERROR_CHK (cyclic redundancy check) and BCC_ERROR_CHK (block check character). A counter indicating the number of packets received with an invalid error check value is maintained. See MVIdf1_HDSGetDiagnostics.

Station sets the station number. The valid range is 0-254. Station number 255 should not be used as it is reserved as a broadcast message designation.

MsgRetries sets the number of times a single message will be transmitted in response to a poll from the master. The range is 1-255. Once this limit is reached, the status of the message is set to “failed”. See MVIdf1_HDSGetPktStat.

Handshake determines whether or not hardware handshaking is enabled and if enabled, the type of handshaking used. Valid values are to be determined.

MVIdf1_HDSOpenPort

RTSOn is the time delay of the transmission of a packet after the RTS output line is activated. This delay only applies when hardware handshaking is active.

Note: If the console is enabled or the Setup jumper is installed, the baud rate for COM1 is set as configured in BIOS Setup and cannot be changed by MVIdf1_HDSOpenPort. MVIdf1_HDSClosePort will return MVI_SUCCESS, but the baud rate will not be affected. It is recommended that the console be disabled in BIOS Setup if COM1 is to be accessed with the DF1 HDS API.

IMPORTANT Once the DF1 port has been opened, MVIdf1_HDSClosePort should always be called before exiting the application.

Return Value:

MVI_SUCCESS	DF1 port was opened successfully
MVI_ERR_REOPEN	DF1 port is already open
MVI_ERR_NODEVICE	UART not found on port
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_OS	Error occurred in DOS 6-XL operating system

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

Example:

```
HDSCFG Port1Cfg;

if (MVIdf1_HDSOpenPort(&PortCfg) != MVI_SUCCESS) {
    printf("Open failed!\n");
} else {
    printf("Open succeeded\n");
}
```

See Also:

MVIdf1_HDSClosePort

MVIdf1_HDSClosePort

Syntax:

int MVIdf1_HDSClosePort (BYTE comport);

Parameters:

comport DF1 port to close

Description:

MVIdf1_HDSClosePort is used by the application to release control of the designated communications port. The application must have previously opened the comport with the MVIdf1_HDSOpenPort API.

IMPORTANT Once the DF1 port has been opened , this function should always be called before exiting the application.

Return Value:

MVI_SUCCESS	DF1 port was closed successfully
MVI_ERR_NOACCESS	DF1 has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
MVIdf1_HDSClosePort(COM1);
```

See Also:

MVIdf1_HDSOpenPort

Communications

MVIdf1_HDSGetPkt

Syntax:

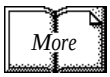
int MVIdf1_HDSGetPkt (BYTE comport, BYTE *DF1_Pkt, WORD *length);

Parameters:

comport	DF1 port from which to fetch a packet.
DF1_Pkt	pointer to the array into which the packet is to be stored.
length	pointer to the variable into which the length of the packet is to be stored.

Description:

MVIdf1_HDSGetPkt determines if a packet is available from the designated port. If available, the application layer data, not the entire DF1 packet, is retrieved.



For more information, refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set.

DF1_Pkt is a pointer to an array where the application layer data will be stored. Only the application layer data will be stored to this array, not the entire DF1 packet.

length is a pointer where the length of the returned packet will be stored.

Return Value:

MVI_SUCCESS	Packet retrieved successfully
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_NODATA	No packet available

Example:

```
BYTE    DF1_Pkt[MAX_DF1_BUFR];
WORD    length;

if (MVIdf1_HDSGetPkt(COM1,DF1_Pkt,&length) == MVI_SUCCESS) {
   printf ("Packet available. \n");
}
```

See Also:

MVIdf1_HDSPutPkt

MVIdf1_HDSPutPkt

Syntax:

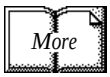
int MVIdf1_HDSPutPkt (BYTE comport, BYTE *DF1_Pkt, WORD *length);

Parameters:

comport	DF1 port from which to fetch a packet.
DF1_Pkt	pointer to array from which the packet is to be retrieved.
length	pointer to variable from which the length of the packet will be retrieved.

Description:

MVIdf1_HDSPutPkt takes the application layer data from the array pointed to by *DF1_Pkt* and places it into the source buffer for transmission. The length of the data is the variable pointed to by *length*. The data passed to this function is only the application layer data, not the entire DF1 packet.



For more information, refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set.

DF1_Pkt is a pointer to an array from which the application layer data will be retrieved. The application should store only the application layer data, not the entire DF1 packet.

length is a pointer to the variable that contains the length of the packet to be stored for transmission.

Return Value:

MVI_SUCCESS	Packet stored successfully for transmission
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
BYTE    DF1_Pkt[MAX_DF1_BUFR];
WORD    length;

if (MVIdf1_HDSPutPkt(COM1,DF1_Pkt,&length) == MVI_SUCCESS) {
   printf ("Packet stored for transmission. \n");

}
```

See Also:

MVIdf1_HDSGetPkt

MVIdf1_HDSGetPktStat

MVIdf1_HDSGetPktStat

Syntax:

int MVIdf1_HDSGetPktStat (BYTE comport, SRCXMT *DF1_Stat);

Parameters:

comport DF1 port on which to request packet status.

DF1_Stat DF1_Stat is a pointer to a structure of type SRCXMT. The SRCXMT structure is defined below in the Description.

Description:

This function returns the status of a packet that has been placed into the source buffer, by the MVIdf1_PutPkt function, for transmission. A transmit status queue is maintained to provide the application with information for each packet placed in the source buffer for transmission.

The transmit status queue contains the status of each packet which has terminated transmission (pass or fail) as well as the status of a packet which may be in the process of being transmitted. The status of packets that have not yet begun transmission will not be in the queue.

Each call to this function returns the status of one packet. Once the status of a packet is reported, that packet's status is removed from the queue and the next query will return the status of the next packet in the status queue.

The last packet status in the queue may be the status of a packet in the process of being transmitted. This packet's status will be returned but it will not be removed from the queue until the query is made when the packet's transmission has been terminated.

The SRCXMT structure is defined below:

```
typedef struct tagSRCXMT
{
        BYTE Src;          // Source node of application layer data
        BYTE Cmd;          // Command code of application layer data
        WORD TNS;          // TNS number for application packet
        BYTE XmitStatus;   // Transmit status of the packet
}SRCXMT;
```

Src the source node value found in the application layer data of the packet

Cmd the command code found in the application layer data of the packet

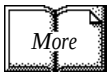
TNS the transaction number found in the application layer data of the packet

MVIdf1_HDSGetPktStat

XmitStatus the status of the packet, where:

MVIDF1_XMITTING	=	packet currently being transmitted
MVIDF1_SUCCESS	=	packet has been successfully transmitted
MVIDF1_FAILED	=	packet transmission failed

Note: The *Src*, *Cmd*, and *TNS* values are to be used by the application to identify the packet to which the status information, *XmitStatus*, pertains.



Please refer to Allen-Bradley publication number 1770-6.5.16, DF1 Protocol and Command Set, for a detailed explanation of the *Src*, *Cmd*, and *TNS* elements of the application layer data.

Return Value:

MVI_SUCCESS	Packet stored successfully for transmission
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)
MVI_ERR_NOSTAT	No packet status available

Example:

```
SRCXMTDF1_Stat;

if (MVIdf1_HDSPutPkt(COM1,&DF1_Stat) == MVI_SUCCESS) {
    printf ("Packet status available. \n");
}
```

See Also:

MVIdf1_HDSPutPkt

MVIdf1_HDSGetDiagnostics

Syntax:

int MVIdf1_HDSGetDiagnostics (BYTE comport, WORD *DF1_Diag, BYTE DF1_DiagNum,BYTE reset);

Parameters:

comport	DF1 port from which to fetch a diagnostic counter value
DF1_Diag	pointer to the variable in which to store the counter value
DF1_DiagNum	number of diagnostic counter to retrieve
reset	reset/no-reset flag for diagnostic counter

Description:

MVIdf1_HDSGetDiagnostics retrieves the value of the designated diagnostic counter. Depending on the value of reset, the counter may or may not be reset to a value of zero.

The diagnostic counters may be used by the application to track and analyze communications problems, monitor packet flow and to allow link optimization.

DF1_Diag is a pointer to a variable in which to store the diagnostic counter value. These counters will roll over to a value of zero if allowed to increment without monitoring and control.

DF1_DiagNum is the number of the desired diagnostic counter value where:

PACKETS_RCVD = 0x00 = Number of valid packets received from the DF1 master.

POLLS_RCVD = 0x01 = Number of polls received from the DF1 master.

NAKS_RCVD = 0x02 = Number of NAKs received from the DF1 master.

BAD_CRC_BCC = 0x03 = Number of packets received with invalid error checks.

DUPS_RCVD = 0x04 = Number of duplicate packets received. This counter is only active if duplicate packet detection is enabled.

PACKETS_XMITTED = 0x05 = Total number of packets transmitted. This value includes message re-transmissions.

SINK_FULL = 0x06 = Number of received packets which have been rejected because the sink (receive) buffer is full. This number indicates that the application may not be retrieving received packets in a timely manner.

MVIdf1_HDSGetDiagnostics

SOURCE_FULL = 0x07 = Number of transmit packets which have been rejected because the source (transmit) buffer is full. This number indicates that the DF1 master may not be polling the DF1 slave in a timely manner.

MESSAGE_RETRIES = 0x08 = Number of packets which have been re-transmitted.

reset determines if the diagnostic counter is or is not reset to a value of zero. A value of non-zero will reset the counter.

Return Value:

MVI_SUCCESS	Packet retrieved successfully
MVI_ERR_NOACCESS	Port has not been opened
MVI_ERR_BADPARAM	Invalid parameter (port number out of range)

Example:

```
WORDDF1_Diag;

if (MVIdf1_HDSGetDiagnostics(COM1,&DF1_Diag,PACKETS_RCVD,0x00) ==
MVI_SUCCESS) {
   printf ("Total number of packets received is %u, \n", DF1_Diag);
// The counter has not been reset
}
```

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 DF1 API User Manual

Cat. No. 1756-MVI Pub. No. 1756-UM008A-EN-P Pub. Date June 2000 Part No. 957234-43

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-UM008A-EN-P - June 2000

PN 957234-43