

**OpenVMS Modbus TCP/IP  
Interface Library  
Programmer's Manual  
Release: 3.1**



Copyright © 2004,2005 Integrated Process Automation and Control Technologies Incorporated

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form without written permission from IPACT Inc.

Technical Writer:

Kevin R. DeYoung  
Staff Engineer  
IPACT Inc.

The following are trademarks of COMPAQ: AXP, VMS. The following are trademarks of Modicon: Modbus, Modbus Plus, SA85, and SQ85. IPACT is a trademark of Integrated Process Automation and Control Technologies Incorporated.

file: W:\Projects\Active\IPACT\MBtcp\Programmers Manual\API\_LIBRARY.DOC  
date: October 20, 2011

<b>Revision Date</b>	<b>Author</b>	<b>Change Description</b>
23-May-2010	E. Lafia	3.0 Support for multiple ports on slave server.
29-nov-2006	E. Lafia	2.8, 2.9 kit updates. Mostly documentation for status and TCP/IP settings.
7-jul-2006	E. Lafia	2.7 kit updates. First Itanium/Integrity release

<b>1</b>	<b>OVERVIEW</b>	<b>1</b>
1.1	Introduction	1
1.2	Interface Library	1
1.3	Available Library Functions	2
1.3.1	Installation	4
1.3.2	Compile Requirements	4
1.3.3	Linking Requirements	5
1.3.4	Performance and Thread Consideration	5
1.3.5	Example Programs	6
1.3.5.1	Process Privileges	6
1.3.5.2	Process Quotas	6
1.4	Modbus Network	7
1.5	PLC Registers and Coil Numbering	7
1.6	PLC MSTR Blocks	7
1.6.1	PLC MSTR Example	8
1.7	Paths	8
1.8	Master Paths	9
1.9	Slave Paths	10
1.10	Route Specifications	12
1.11	Non PLC Modbus Nodes	12
1.12	Host and PLC Data Byte Order	12
1.13	Application Status Returns	13
1.14	Contention and Synchronization Issues	13
1.15	Process Expanded Region	14
1.16	Function Status Codes	14
1.17	The Concept Of Timeouts	14
1.18	Unsolicited Message Facility	15
<b>2</b>	<b>LOCAL HOST SLAVE SERVER</b>	<b>17</b>
2.1	Introduction	18
2.2	Slave Server Description	18

2.3	Slave Server Connections	19
2.4	Slave Server Startup	20
2.5	Slave Server Holding Registers	20
2.6	Direct access to the Virtual PLC (Holding Registers)	21
2.7	Unsolicited Messaging	21
<b>3</b>	<b>MBP APPLICATION LIBRARY CALLS</b>	<b>23</b>
3.1	MBP_CLOSE_NET	24
3.2	MBP_CRE_SIGNAL_OBJ	25
3.3	MBP_DEL_SIGNAL_OBJ	26
3.4	MBP_FORCE_SINGLE_COIL(W)	27
3.5	MBP_MAP_VPLC	30
3.6	MBP_OPEN_NET	32
3.7	MBP_PRESET_SINGLE_REGISTER[W]	34
3.8	MBP_READ_EXTENDED[W]	37
3.9	MBP_READ_REGISTERS[W]	41
3.10	MBP_READ_UN SOLICITED	45
3.11	MBP_REGISTER_UN SOLICITED	48
3.12	MBP_RESUME_UN SOLICITED	52
3.13	MBP_SUSPEND_UN SOLICITED	54
3.14	MBP_VPLC_ACCESS	56
3.15	MBP_WRITE_EXTENDED[W]	58
3.16	MBP_WRITE_REGISTERS	62
<b>4</b>	<b>EXAMPLE PROGRAMS</b>	<b>66</b>
4.1	Introduction Test Programs	67
<b>5</b>	<b>UTILITIES</b>	<b>69</b>
5.1	READREG DCL Utility	70
5.1.1	ReadReg Command Syntax	70

5.1.2	ReadReg Examples	72
<b>5.2</b>	<b>WRITEREG DCL Utility</b>	<b>74</b>
5.2.1	WriteReg Command Syntax	74
5.2.2	WriteReg Example	75
<b>6</b>	<b>HEADER FILES</b>	<b>76</b>
6.1	“C” Header Files	77
6.2	Header Files	77
<b>7</b>	<b>MBP_ERROR CODES</b>	<b>78</b>
7.1	API Error Status Returns	79
7.2	Modbus Exception Codes	79
7.3	MBP Error Codes	81
<b>8</b>	<b>APPENDIX A SAMPLE PLC MSTR</b>	<b>87</b>
8.1	MSTR Example	88
<b>9</b>	<b>APPENDIX B EXAMPLE VMSINSTALL</b>	<b>93</b>
9.1	Example Product Install	94
<b>10</b>	<b>APPENDIX C IFIX ACCESS TO VIRTUAL PLC</b>	<b>97</b>
10.1	Setting up the datablocks in iFIX	98
10.2	Selecting max number of sockets to use	100
<b>11</b>	<b>APPENDIX D IFIX MODBUS TAG EXPORT</b>	<b>101</b>
11.1	Exporting Modbus tags from iFIX to a CSV file	102
<b>12</b>	<b>APPENDIX E WONDERWARE INTOUCH &amp; VIRTUAL PLC</b>	<b>106</b>
12.1	Configuring the Access Name	107
12.2	Configuring the MBENET I/O Server	107
<b>13</b>	<b>APPENDIX F INTOUCH MODBUS TAG EXPORT</b>	<b>109</b>
13.1	Configuring the Access Name	110

<b>14</b>	<b>APPENDIX G TCP COMMUNICATION ISSUES _____</b>	<b>111</b>
14.1	TCP Communication Configuration	112
<b>15</b>	<b>APPENDIX H MODBUS OVER TCP/IP COMPLIANT DEVICES _____</b>	<b>114</b>
<b>15.1</b>	<b>Compliant Devices</b>	<b>115</b>
15.1.1	GE FANUC PAC processors	115
15.1.2	Rosemont FIM 3420	115
15.1.3	Thermo-Fisher X-ray	115

# **1 Overview**

# OVERVIEW

## Introduction

---

### 1.1 Introduction

This document describes the Modbus© TCP/IP Application Interface Library (API). It describes in detail, the operation of each supported function. It is assumed that the user of the API is familiar with Modbus terminology and operation. There are a number of example programs supplied with the API kit to allow thorough testing of all aspects of the kit software. The document also provides some integration help in using this software along with some actual PLC rungs that provide examples of communication logic operations.

### 1.2 Interface Library

The Modbus TCP/IP OpenVMS interface library provides a set of callable routines which reduce the effort required to communicate with Modbus devices accessible via ethernet over TCP/IP. The Modbus over TCP/IP is an open protocol which is supported by many vendors (see: [www.modbus-ida.org](http://www.modbus-ida.org)). The API is implemented as an object library that is linked into each application requiring access to the target Modbus devices.

The API software assumes the presence of a TCP/IP protocol stack on the OpenVMS host system. The API software uses standard “C” socket interface routine calls in its implementation for maximum portability. The API also uses POSIX Compliant Threads calls in its implementation.

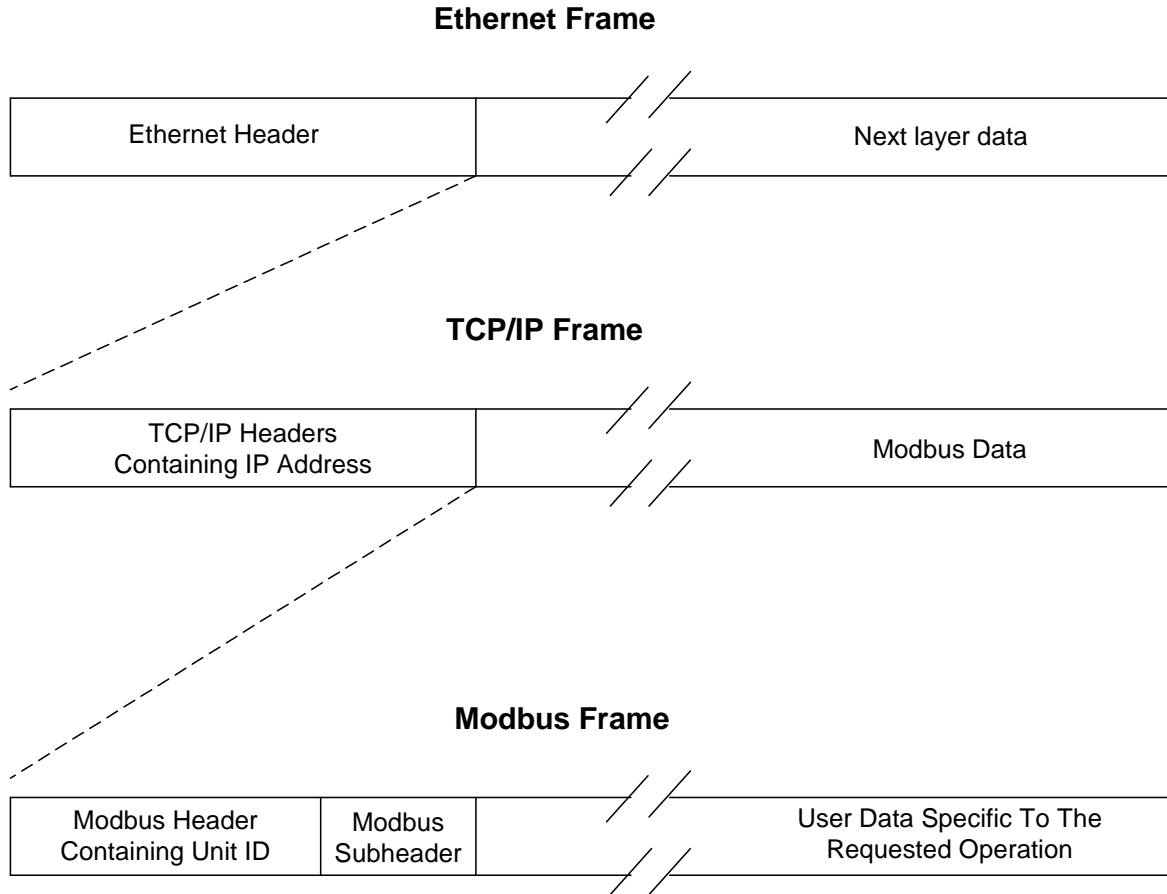
Users of the API library should familiarize themselves with the following Modicon documents: “[Modicon Ladder Logic Block Library User Guide](#)”, and the “[Open Modbus/TCP Specification](#)”. Both of these are available through Modicon. The “[Open Modbus/TCP Specification](#)” is available through the Modicon website <http://www.modicon.com/openmbus> in either HTML or WORD format.

The Open Modbus/TCP specification is a description of how Modicon has layered its Modbus protocol on top of standard TCP/IP. The method of node addressing is the primary change in the packet format from Modbus to Modbus/TCP. The Modbus protocol maintains a 5 byte “Route” list as part of each packet moving about the Modbus Highway. The Route list contains the single byte Modbus address of each Modbus entity between the source node and the destination node of the packet. With Modbus/TCP, the participating nodes in a conversation each have a 5 byte route array as well. The difference is in how the bytes are used. With Modbus/TCP, the first 4 bytes of the route array are actually the TCP/IP address of the destination node, or PLC. When an API call issues a request to a node, the first four bytes of the route array are included in the standard TCP/IP frame header; this is part of the TCP/IP standard. The fifth and final byte of the Modbus/TCP route array becomes the Unit Identifier in the imbedded Modbus frame. The following diagram shows how the frames are structured.

## OVERVIEW

### Available Library Functions

---



All of the functions supplied as part of this API are base upon the concept of a Path. A path in the context of Modbus/TCP is a virtual entity rather than a physical one. A path is nothing more than a data structure created to track connections with remote Modbus devices. The structures are initially created by the MBP\_OPEN\_NET function. The number of paths of each type a specified at this time. All other API calls require a path specification to be supplied. When a command is issued to a Modbus device for the first time, the association between the device and the supplied path is made. See section Paths for further information on the topic of paths.

### 1.3 Available Library Functions

The MBP interface calls and error status messages are all prefixed with "MBP\_". Many of the functions are available in either synchronous or asynchronous versions. These are signified by a trailing (W) indication. The Modbus/TCP library contains the following functions:

**MBP\_CANCEL\_OP-** Cancel an outstanding Modbus operation active on a given path.

**MBP\_CLOSE\_NET-** Close all links and deallocate all memory which was allocated as part of the MBP\_OPEN\_NET call.

## **OVERVIEW**

### **Available Library Functions**

---

**MBP\_CRE\_SIGNAL\_OBJECT**- Creates a synchronization object used in signalling the user application upon completion of an operation.

**MBP\_DEL\_SIGNAL\_OBJECT**- Deletes a synchronization object which was previously created by a call to the **MBP\_CRE\_SIGNAL\_OBJECT** function.

**MBP\_FORCE\_SINGLE\_COIL(W)**- Force a single coil within a PLC to a specific state.

**MBP\_MAP\_VPLC**- Maps the virtual PLC (slave server holding registers).

**MBP\_MASK\_WRITE\_REGISTER(W)**- Performs a masked write operation on a specified PLC holding register as described in the OPEN Modbus/TCP Specification.

**MBP\_OPEN\_NET**- Creates the required data structures and threads to support future calls to the API functions..

**MBP\_PRESET\_SINGLE\_REGISTER(W)**- Set a single holding register within a PLC to a specific value.

**MBP\_READ\_EXCEPTION\_STATUS(W)**- Reads the exception status value from a target PLC.

**MBP\_READ\_EXTENDED(W)**- Read extended register files from a specified PLC. This function is referred to as `read_general_reference` in the OPEN Modbus/TCP Specification.

**MBP\_READ\_REGISTERS(W)**- Read a consecutive series of registers or coils from a PLC. Only holding registers are supported when the target device is another Host system. The operation performed is based upon the address range specified.

**MBP\_READ\_UNSOLICITED**- De-queue a message from the specified unsolicited message mailbox.

**MBP\_REGISTER\_UNSOLICITED**- Register for unsolicited slave messages from PLCs or other Modbus/TCP hosts which are accessible via ethernet. Allocate a mailbox and initiate reception of Master messages to the local host Slave Server process on a specified Slave Path.

**MBP\_RESUME\_UNSOLICITED**- Resume transmission of previously suspended, unsolicited messages to the calling application from the local host Slave Server on a specified path.

**MBP\_SUSPEND\_UNSOLICITED**- Suspend transmission of unsolicited messages from the local host Slave Server process to this application on a specified path.

**MBP\_VPLC\_ACCESS**- Locks access to the virtual PLC (slave server holding registers). Applications should use this to ensure that virtual PLC reads and writes are protected such that the data is not read in the middle of a multi-register update.

## OVERVIEW

### Available Library Functions

---

MBP\_WAIT\_SIGNAL- Wait for completion of a Modbus transaction based upon a signal object previously created by the MBP\_CRE\_SIGNAL\_OBJ call.

MBP\_WRITE\_EXTENDED(W)- Write registers to extended memory files in a PLC. Also referred to as write\_general\_reference in the OPEN Modbus/TCP Specification.

MBP\_WRITE\_REGISTERS(W)- Write a consecutive series of registers or coils in a PLC or a Modbus device. Only holding registers are supported by Host systems. The operation performed is based upon the address range specified.

Each function above is documented in chapter: 3. Most functions may be called as either synchronous or asynchronous. Asynchronous calls return to the caller immediately with an indication of whether the supplied parameters were acceptable. I/O completion is signaled via a signal object and final status is returned via the user supplied status block. The synchronous version of the calls, the [W] variant, returns to the caller upon completion of the operation. (e.g. MBP\_WRITE\_REGISTERSW). Only a single operation may be active on a given path at a time. This is due to the design of the Modbus/TCP API. However, each defined path may have an operation in progress at the same time, giving the user a degree of parallelism.

#### **1.3.1 Installation**

The API software library is installed via VMSINSTAL. A sample VMSINSTAL session is included in the appendix A. After installation, the logical "MBT\_" will point to the directory of the installed software. The command procedure "MBT\_STARTUP.COM" should be added to the system or application startup command procedure. MBT\_STARTUP.COM will define the logical names required for accessing the library files required to build the user applications. The VMSINSTAL kit uses the current version and revision number of the API library to create a unique software distribution directory (e.g. SYS\$COMMON:[MBT010]). The software is placed in the root of SYS\$COMMON or SYS\$SYSDEVICE.

#### **1.3.2 Compile Requirements**

The following text libraries are available in the library directory. These libraries contain the Modbus message definitions, and structures required for accessing the Modbus/TCP application library and the data returned by them. These libraries are:

MBPDEF\_C.TLB- C header definitions and function prototypes  
MBPDEF\_F.TLB- FORTRAN structure and function definitions

The following are contained in these libraries:

- MBP\_BUFDEF- Definition of structures and messages exchanged in transactions between the application and Modbus devices.
- MBPDEF- Contains the function prototypes for all of the MBP\_ function calls.
- MBPPEX- Definition of the process expanded region created by the MBP\_OPEN\_NET function.

## OVERVIEW

### Available Library Functions

---

- MBP\_PEXDEF- Definition of all structures and types used by the API library external and internal functions.
- MBPSHARE\_MSG- Definition of all error codes returned by the functions contained in the API library.

For “C” programmers, the user should compile and link with the following command (assuming source module is: “test.c”):

```
$CC test+mbt$lib:mbpdef_c.tlb/library
```

For FORTRAN programmers the user may specify the location of the text library in the include statement within the source module.

### **1.3.3 Linking Requirements**

The user has a choice of linking to either a shared library or an object library. To link to the object library the linker commands are:

```
$LINK/EXE=test.exe SYS$INPUT/OPTIONS  
test  
mbt$lib:mbt_lib/include=mbtshare_msg  
mbt$lib:mbtlib.olb/library
```

To link to the shared library, the following linker commands should be used:

```
$LINK/EXE=test.exe SYS$INPUT/OPTIONS  
test  
mbt$lib:mbt_library_shr/share
```

The logical "MBT\_" and mbt\$lib are defined by the command procedure, MBT\_STARTUP.COM, created by VMSINSTAL kit.

The object library method is provided to allow the programmer to link to an object library or a debug object library. The debug library allows the programmer to isolate faults that might be occurring within the shared library, and assist in reporting bugs to the developers. These two libraries are: MBTLIB.OLB and MBTLIB\_DBG.OLB.. The debug version of library should be used only with support from IPACT. Additionally, the debug library should not be linked with the linker option: “/thread\_enable”.

### **1.3.4 Performance and Thread Consideration**

The MBTLIB routines use POSIX threads. Maximum throughput can not be achieved unless threads are enabled. This can be done using the following addition to the linker command line as follows:

## OVERVIEW

### Available Library Functions

---

```
$LINK/EXE=test.exe/THREADS_ENABLE SYS$INPUT/OPTIONS
```

```
test
mbt$lib:mbt_lib/include=mbtshare_msg
mbt$lib:mbtlib.olb/library
```

It is also possible to change or view a processes thread setting using the THREADCP DCL command (note, this command is not normally available and may require the "\$set command sys\$update:threadcp.cld").

```
$THREADCP/show test.exe
```

```
$THREADCP/THREADS_ENABLE test.exe
```

For example:

```
$ threadcp/show TEST.EXE
TEST.EXE;1
%THREADCP-I-MKT, multiple kernel threads are enabled
%THREADCP-I-UPC, upcalls are enabled
```

In addition the VMS sysgen parameter: MULTITHREAD should be set to one or greater. The change in performance without threads are a power of ten faster.

### **1.3.5 Example Programs**

The subdirectory "MBT\$SRC:[.Examples]" contains example programs that may be used to test the network or, as a template to be used in designing user applications. ***The user is cautioned that these example programs will write to the target Modbus node causing undesirable results if the target node is controlling a process.***

#### **1.3.5.1 Process Privileges**

Users of the Modbus/TCP API must have the following privileges:

SYSNAM- To allocate a permanent mailbox for data messages from the PLCs.  
GRPNAM- To allow shared access to the named global section.

#### **1.3.5.2 Process Quotas**

This software does not require any abnormal quotas to function. However, the following are provided for fault analysis, and determining if normal quotas are not present.

PGFLQUOTA- Page file quota. The process-expanded region is allocated from the system page file. This is typically three pages plus one page for each path that is allocated.

WSQUOTA- See PGFLQUOTA

## **OVERVIEW**

### **Modbus Network**

---

#### **1.4 Modbus Network**

The Modbus/TCP API library calls are very similar in nature to the function calls found in the VMS Modbus Plus Interface Library, which has been available from IPACT, Inc. for a number of years. There are some basic differences in the concept of paths and routes between these interface libraries. The concept of a path in the Modbus/TCP API library is one of a virtual nature, while in the VMS Modbus Plus library it was a physical attribute of the controller hardware. The route used in the VMS Modbus Plus library was an actual Modbus route which specified how to locate the destination device. The Modbus/TCP library uses a similar route structure of 5 bytes, but the contents are actually the TCP/IP address bytes plus a Modbus unit identifier byte.

#### **1.5 PLC Registers and Coil Numbering**

All of the calls provided by the Modbus/TCP API library use the same numbering system as the actual PLC itself. The user need not bias or normalize any of the registers or coil numbers before passing the register address to the library function. This allows the PLC programmer and the host programmer the ability to converse using identical register and coil numbers. The software examines the address, such as 40001, and knows that this is the first holding register, or 00001 is the first coil status.

#### **1.6 PLC MSTR Blocks**

The PLC's MSTR block is the mechanism used by Modicon PLCs, to communicate with other Modicon devices or hosts. When using the MSTR Instruction the PLC is the Master device and the target of the communication is the Slave.

The PLC MSTR instruction requires a routing path, and a Modbus Plus function code (along with other parameters). The routing path specifies the target slave node that can be another PLC or a host computer with a Slave Server process running on it. The PLC programmer must be aware of the differences in the use of Control Block Registers when communicating with a Modbus connected device versus an ethernet connected device. These differences are spelled out clearly in the "[Modicon Ladder Logic Block Library User Guide](#)". The section titled "The Modbus Plus Master Instruction" has complete listings of Control Block register usage.

The preferred method for mapping the slave data paths and Host computer node addresses is to have the Host computer specify these parameters to the PLC in a set of predefined holding registers, directly. By using this methodology, the PLC does not need to maintain this part of the link management. This has the following benefits:

- The programmer of the PLC need not worry about communicating changes to the programmer of the Host computer unless agreed upon locations of the link management registers are to be changed.
- The ability to develop redundant Hosts without changing logic in the PLC.

## OVERVIEW

### Paths

---

- The ability for the Host programmer to partition and even distribute functionality to multiple programs running on either the same or different Hosts.

#### 1.6.1 PLC MSTR Example

Assuming that there are eight PLCs and two OpenVMS Hosts. Each of the PLCs has two MSTR blocks. The first MSTR Instruction is responsible for sending a transaction containing completed counts and statistics about a product just made to the "Production Management" Host. The second MSTR Instruction is responsible for uploading significant event and current process history to a process "Process History" Host. In the event that either of the Hosts are unavailable, the other Host assumes the functions of the failed Host.

The Host which assumes responsibility for either, or both, of the supervisory functions would write its TCP/IP Route information to the control area registers for the required MSTR Instructions in each PLC. If the Host is performing just one of the supervisory functions it would fill in only the appropriate MSTR Instruction Control Registers on each PLC, with its Route information. If the Host is assuming both aspects of the supervisory operation it would perform the same operation for both MSTR Instructions on each PLC.

A sample PLC program is shown in the appendix.

#### 1.7 Paths

Communication between a Host computer and an ethernet capable Modicon device occurs over a standard TCP/IP connection in the Modbus/TCP API library. The concept of a Path is changed somewhat from the original VMS Modbus Plus Interface Library which was limited by the capability of the interface card. The Modbus/TCP API library uses both Master and Slave paths to carry out operations with Modicon devices. The use of a Master Path versus a Slave Path is dictated by the operation desired by the author of the Host application. Master Paths are used for initiating transactions where the Host computer is the Master device. A Modicon PLC or another Host computer might be the target, or slave device, for the specified transaction. The Slave Paths are used exclusively for operations where the local Host computer is the Slave in the transaction. Each case will be discussed in subsequent sections.

The paths allocated by the MBP\_OPEN\_NET are allocated from a free pool of memory. The MBPPEX structure returned to the caller of MBP\_OPEN\_NET is the only connection the application has to the expanded region of memory. If the application disturbs the contents of the MBPPEX structure, unpredictable operation or total application failure will result.

All the routines documented in this manual which specify a path, reference the path by its index (1 through N). All functions except the calls related to Unsolicited data, refer to a Master Path. It is irrelevant which path is used in communicating with a slave device except where parallel operation is to be considered. If your application can improve its performance by issuing simultaneous commands to multiple devices, one should group devices on different paths. Once device "A" has been accessed via path 1, you should

## **OVERVIEW**

### **Master Paths**

---

continue to use path 1 for device “A”. If performance can be improved by accessing both device “A” and “B” simultaneously, then device “B” should be accessed on a different path. In theory, there is a limit to the number of paths, which may be specified in the MBP\_OPEN\_NET function call.

Due to the signaling mechanism in OpenVMS, the current version is limited to a total of 16 paths. This means that the total number of Slave and Master Paths specified in the MBP\_OPEN\_NET call may not exceed this number. In turn, each Master Path specified can maintain as many as 32 links, or connections, on a single path. Each link is then a connection to a Modbus device on the Ethernet. These limits are per process. The Local Host System may have many MBTCP based applications running on the same node, each with as many as 16 paths and 32 links for each path. The exception to this rule is the Slave Server application.

Theoretically, there may be one MBT\_SLAVE\_SERVER process per Ethernet connection on the host computer. Some implementations support alternate ports, which would allow for additional copies of the process on each Ethernet connection. This solution would be a non-standard configuration. If the host computer has two Ethernet controllers, each with a unique TCP/IP network ID, the host could support two MBT\_SLAVE\_SERVER processes. Currently, this support has not been added to the MBT\_SLAVE\_SERVER process. IPACT may consider this option in the future if the need arises.

### **1.8 Master Paths**

A Master Path is used when the Host computer is the “Master” and the PLC or other remote Host or Modicon device is the “Slave”. A Master Path in the Modbus/TCP API architecture is simply a mechanism by which the interface is able to allow Modbus operations to be initiated. A single application is able to have many Master Paths available to it, each of which may have a transaction in progress simultaneously. The author of an application may wish to issue commands to multiple PLCs simultaneously. This is accomplished by issuing commands to the target PLCs via different Master Paths. An association between a specific slave PLC and a given Master Path on the Host computer will then exist. The application may also issue commands to multiple slave PLCs via the same Master Path. Only one operation, or transaction, may be active on a single Master Path at a time. Thus, you may communicate with multiple slave devices via a single path, you must do so in a serial fashion.

## **OVERVIEW**

### **Slave Paths**

---

#### **1.9 Slave Paths**

Slave Paths, in the Modbus/TCP architecture, are used exclusively in communication with a local Host Slave Server process. The Slave Path may not be used by function calls other than those related to Unsolicited data reception. The local Host Slave Server process enables the Host computer to receive commands from Modicon devices acting as Masters or from other Host computers, which are issuing Master commands to the local Host. In effect, the Host computer emulates a Modicon PLC. The software has been tested with iFix by Intellution and InTouch by Wonderware HMI software.

MBP\_OPEN\_NET allocates available slave paths in sequential order. A slave path is used when the local Host is the slave and a PLC or another Modbus node, or Host computer is the master. The PLC uses the MSTR Instruction to read from or write to the local Host computer. If the MSTR on the PLC initiates a write operation to the local Host, the Slave Server process receives the information and writes it to memory locally.

If user applications have registered for unsolicited data, the message from the Master, plus an additional header are delivered to the registered application via a mailbox (see MBP\_REGISTER\_UN SOLICITED for further details).

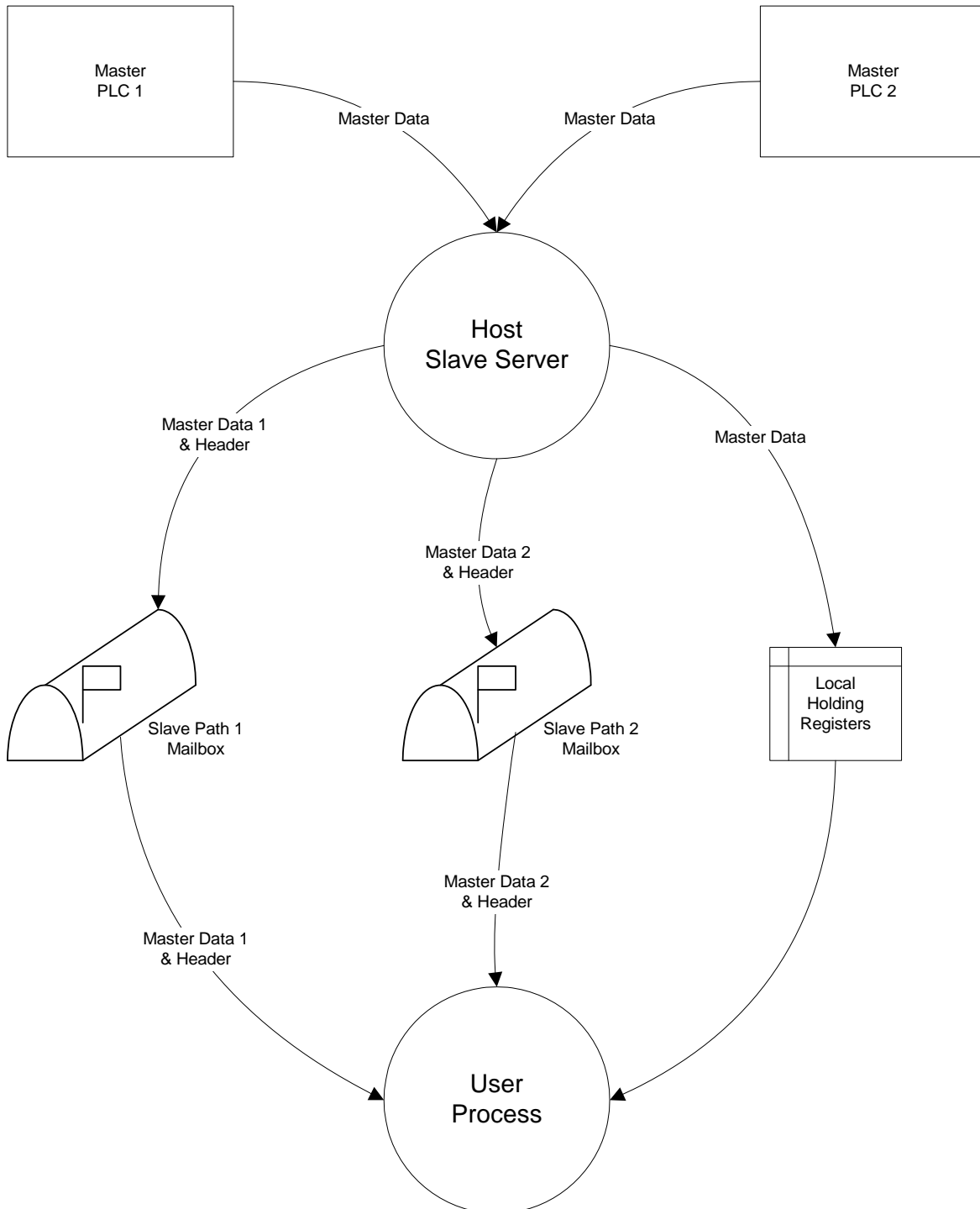
This is illustrated in the following diagram.

# OVERVIEW

## Slave Paths

---

### FLOW OF UNSOLICITED DATA



In the previous diagram the user's application has chosen to register for unsolicited data on two different slave paths. The user has specified a single Master route specification for each slave path as well. The effect of this method is to have the Slave Server deliver

## **OVERVIEW**

### **Route Specifications**

---

all data from a PLC1 via the mailbox associated with slave path 1 and all data from PLC2 via the mailbox associated with slave path 2. The PLC1 and PLC2 devices could also have been an HMI or other type of HMI. The user could have supplied a wildcard type of route specification and had data from both PLC 1 and PLC 2 delivered via a single mailbox through one slave path. This feature is discussed in more detail in the chapter on the Slave Server operation.

#### **1.10 Route Specifications**

Although the concept of a route may seem similar between the Modbus/TCP environment and its predecessor, Modbus Plus VMS Interface, it is very different. In the case of the Modbus/TCP environment, there are five bytes used for the “route” information. The first four of five are actually the TCP/IP address bytes of the target device or system. These are normally written as aaa.bbb.ccc.ddd, where aaa, bbb, ccc, and ddd are decimal numbers in the range of 0 – 255. The fifth byte is a “unit” identifier which is also a decimal number in the range of 0 – 255. The unit identification comes into play when a bridge is used as the connection between the ethernet and the Modbus highway. The unit identifier can then be used as an index into the devices routing table. See the user’s guide for the specific device for further information on the use of the value.

#### **1.11 Non PLC Modbus Nodes**

It is possible to have Modbus Nodes which are not programmable Controllers. Typical non-Modbus nodes are either other Host Computers, or operator displays. A typical application is where the Host Computer provides supervisory information such as order information, customer names, product descriptions, and other data not needed for the actual operation or control of the operations. IPACT has used this feature with both UNICEL© marketed by Modicon, Factory Link© by USData, and InTouch© by Wonderware. Operator screens can be made that display information seamlessly without regard to the source of the information.

#### **1.12 Host and PLC Data Byte Order**

The Modicon PLCs and the typical Host computer, store data bytes in different byte order. For data read from, or written to, registers in the PLC, the API software will automatically byte swap the register data. For example, if a holding register in the PLC contained a simple counter and it currently read "2011" (hex), the same sixteen bit data word read by the Host computer would have been "1120" (hex) if this software did not swap the data. This byte swapping is only performed on register data (MBP\_READ\_REGISTERS, MBP\_WRITE\_REGISTERS, MBP\_WRITE\_EXTENDED, MBP\_READ\_EXTENDED). If the application requires the data to be delivered in its natural byte order, a flag is used in these function calls to achieve the desired result.

Data which is received via the “Unsolicited” mechanism is left in its natural byte order as found on the source device. It is the responsibility of the user application to properly interpret the data received via this mechanism.



## **OVERVIEW**

### **Process Expanded Region**

---

#### **1.15 Process Expanded Region**

Each process that uses the Modbus/TCP API library has a group global section created when MBP\_OPEN\_NET is called. This region is used by the Modbus/TCP threads to coordinate activities on the various paths. The region is given a unique name, which includes a portion of its process identification. The name of the region is constructed by appending the process id to the string "MBPREGxxxxxxxx" where xxxxxxxx is the process identifier for the application, which called MBP\_OPEN\_NET.

#### **1.16 Function Status Codes**

The successful status codes returned by all Modbus/TCP API functions on OpenVMS systems are odd numbered. All status codes indicating failure, on OpenVMS systems, will be even. This is standard for OpenVMS based systems and somewhat non-standard for Unix based systems. Many of the API calls will actually return the operating system service status in cases of failure. If the cause of failure is for a reason specific to the Modbus/TCP environment, one of the codes listed in the MBPSHARE\_MSG include file will be returned. It is generally best to compare the return code with MBP\_SUCCESS.

#### **1.17 The Concept Of Timeouts**

The concept of a timeout in the Modbus/TCP environment requires some explanation before moving to the interface calls themselves. There are no "Master" functions which are open ended, or in other words, which are used for handling of unsolicited information from Modbus/TCP "Slave" devices. All "Master" operations are two part operations. Each operation is handled by a thread, which was created as part of the MBP\_OPEN\_NET function call. The first part of each operation is a request from the local Master, the application in this case, to a specific "Slave" device or system. The second part is the response of the "Slave" to the "Master" request. The time elapsed between initiating the request and the arrival of the response will vary depending upon three main factors.

- The complexity of the ethernet/TCP network to which the Modbus/TCP devices are connected.
- Work load of the slave device or system (PLC or Host system).
- Work load of the local Host system.

The amount of time required for a message to propagate from the local Host system to a target Modbus device, may vary greatly based upon the underlying network complexity. Since TCP/IP was designed for use in Wide Area Networks, remote targets could be on the same local ethernet segment, or they might be located at a different manufacturing site hundreds of miles away. Even local ethernet networks may have significant delays, depending on the type of network design and the number of subnets or routers between the Master and Slave devices.

If the target device is a PLC, there may be additional delays due to the primary control functions of the PLC. A very busy PLC may respond rather slowly when compared to a lightly loaded PLC of a similar type.

## **OVERVIEW**

### **Unsolicited Message Facility**

---

The same is true of a Modbus/TCP Host system. Its normal workload may have been given priority over the Modbus/TCP environment of the user application. All of these factors must be evaluated when considering timeout values.

The synchronous and asynchronous versions of the Modbus/TCP function calls require a timeout specification. In the case of the asynchronous variant of the function calls, the timeout value specifies how long a thread should wait around for a response from the "Slave" before moving onto the next work opportunity. Although the application is free to continue its processing immediately following the function call, the thread which is handling the operation will wait as long as "tmo" milliseconds before giving up on the requested operation response. If unusually long timeout values are given, the I/O thread could be unavailable for long periods of time. On a reasonably designed ethernet with normally loaded target devices, responses would be measured in hundreds of milliseconds.

In the case of the synchronous variant of the calls, the timeout value is the time which the application is willing to wait before declaring a failure. In this case the timeout value, in milliseconds, is also used by the thread which handles the underlying I/O to the slave. However, in the synchronous variant, control is not returned to the calling application until the timeout period has expired. If the request requires multiple frames, each frame uses the timeout until all frames are processed or until a timeout occurs.

It should be noted that `mbp_open_net` does not create an actual socket to any target node. It is not until the first attempt to access the target nodes that the TCP/IP connect is attempted. Therefore, it may seem that the timeout function is not working. A connect timeout error will be reported instead after the system TCP/IP tuning parameter for the connect is exceeded. Typical OpenVMS values are seventy-five (75) seconds for this parameter. A discussion of TCP/IP parameter settings are listed in section: 14.1

### **1.18 Unsolicited Message Facility**

The Unsolicited Message Facility supported by the Modbus/TCP API allows the user application to directly receive Modbus messages sent to a local Host Slave Server. The Slave Server is a local host process which emulates certain aspects of a Modbus PLC. The Slave Server supports a user specified number of local holding registers (4xxxx register range). Any master device supporting Modbus/TCP may issue read and write requests for the defined Slave Server holding registers just as if they were resident on a real Modbus PLC. An application on the local host may also issue read and write requests for these same holding registers.

When the Slave Server receives a command to write data into the local holding registers, it checks to see if a user application has registered an interest in writes from that source. If interest has been registered for the source node, a copy of the write request message is forwarded to all interested applications. The message, as defined in the Open Modbus/TCP Specification, is inserted into a structure containing a command/status longword, and a copy of the source node's route information, and is then sent to interested applications. The data contained in the message is in its original form including the source's local byte order. See the chapter on the Slave Server process for a complete description of Slave Server messaging facility.

## **OVERVIEW**

### **Unsolicited Message Facility**

---

When mapping the virtual PLC (slave server holding registers) directly, updates to the virtual PLC's holding registers will not trigger unsolicited messages to be sent. Keep this in mind when designing your applications.

## **2 Local Host Slave Server**

## **Local Host Slave Server**

### **Introduction**

---

#### **2.1 Introduction**

A Slave Server is a process running on the local Host system that emulates the holding registers of a Modicon PLC. The Slave Server supports only the emulation of a user specified number of holding registers in the 4xxxx range. This is also referred to as the virtual PLC in this document. Modbus devices and host systems supporting the Open Modbus/TCP specification may perform read and write operations on the local Slave Server holding registers, as if they were resident in a normal Modbus PLC (in effect, a Virtual PLC on OpenVMS without the ability to solve ladders).

The Slave Server is typically used for testing when the real Modicon PLC or other Modbus over TCP/IP device is not available or for integration of HMIs. HMI products like Intellution iFIX, Wonderware InTouch, and others that support the open Modbus TCP/IP protocol will be able to read and write to the holding registers contained in the group global section supported by the Slave Server. An example configuration for iFix is shown in section Setting up the datablocks in iFIX.

In addition to this limited holding register functionality, the Slave server also supports the Unsolicited Message Facility. When a write command is received by the local VMS Host Slave Server, the function is performed upon the designated local holding registers. Following the update of the local holding registers, a copy of the Master's write command message is sent to all applications which have registered an interest in writes from the Master device which issued the command. The user must be aware that there is no guarantee of command delivery over TCP/IP and this API. System (mailbox overflows) and network problems may result in the loss of a command from a master to the local slave. There is also no guarantee of message delivery from the Slave Server to local applications which have registered for unsolicited message delivery, due to the volatile nature of the mailbox facility under OpenVMS. The application designers on the Level 1 and the OpenVMS end must be aware of this in their design. However, this limitation is similar to the limitations imposed by any typical host and PLC design.

The most common application is the merging of the OpenVMS data and the PLC data on an HMI in a cohesive manner. The messaging support is commonly used for asynchronous messages from a PLC (such as a tracking update based on a photo eye or other detection device) or on demand updates from an HMI. However, most HMI applications poll or scan the PLC and this could result in a message each HMI refresh. The user should simply understand how outputs and inputs are implemented by the HMI to determine which methods are used.

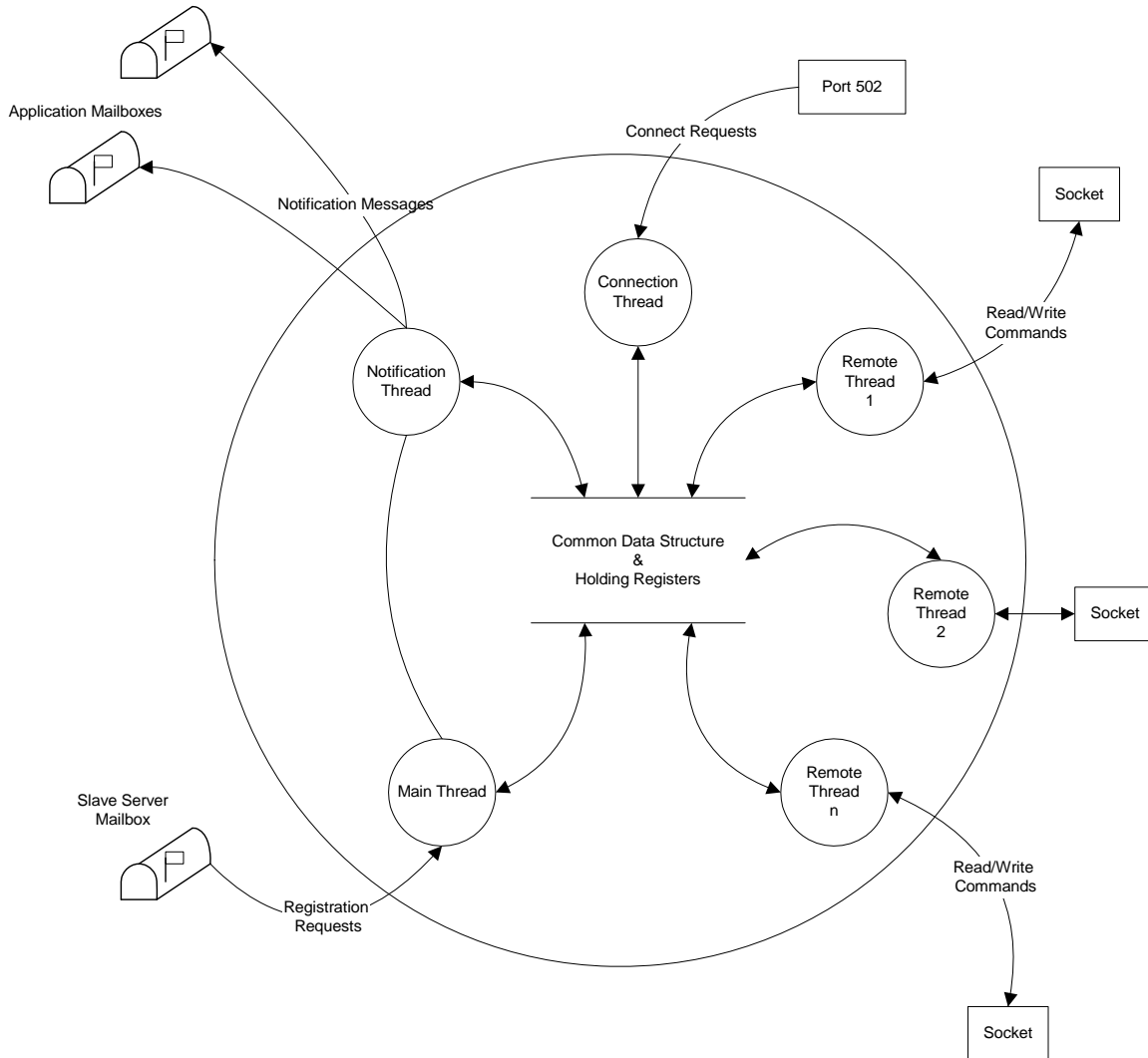
#### **2.2 Slave Server Description**

The actual Slave Server process is a collection of threads which share a common data structure. Part of that data structure is the holding registers. The following diagram illustrates the design of the Slave Server.

# Local Host Slave Server

## Slave Server Connections

### SLAVE SERVER PROCESS



Unsolicited message registration and delivery is handled through mailboxes. The Slave Server has its own permanent mailbox through which it receives requests for registration, de-registration, and message suspend/resume requests. The Main Thread handles all the unsolicited requests. Any application issuing a request of the Slave Server for an action regarding unsolicited messages, will receive an acknowledge message via its mailbox its own mailbox, for the requested operation.

### 2.3 Slave Server Connections

The Slave Server listens on socket 502 for connections. Each external system that wishes to read and/or write data will connect through the modbus protocol to that socket. A new socket will be created by the slave server to support the communications with the external system. Should the network connection to that external system be

## **Local Host Slave Server**

### **Slave Server Startup**

---

disconnected (via network cable disconnect, via system crash of the external system), the Slave Server uses the TCP keepalive feature to time out broken connections. In absence of messages on the socket, TCP will send keepalive probes, and if they are not responded to by the external system's TCP stack, the connection is deemed broken and socket will be destroyed.

#### **2.4 Slave Server Startup**

The Slave Server must be started with a DCL context (e.g., Batch Job or using sys\$system:loginout.exe). The Server accepts parameters from the command line as shown below. The "Slave" foreign symbol must be defined as: "\$mbt\$prod:mbt\_slave\_server.exe". Where the "mbt\$prod" is the product production directory for this product defined by the "SYS\$STARTUP:MBT\_STARTUP.COM" command procedure created by the kit installation process.

#### **\$ SLAVE VPLC\_NAME [StartRegister RegisterCount LocalPaths SlavePaths]**

**VPLC\_NAME** = Name for the PLC. This becomes the name of the group global section which can be mapped by the MBP\_MAP\_VPLC call "REGION\_NAME" parameter (see: MBP\_MAP\_VPLC).

**StartRegister**= If not specified, then 40001 is the first 16 bit value in the global region and first holding register served by the Slave Server. Readers unfamiliar with Modicon PLCs, the first holding register is configurable. IPACT recommends that 40001 always be used.

**RegCount**- Number of 16 bit holding registers to support. Default is 1000.

**LocalPaths**- Number of local paths to support. One local path is consumed for each process that desires notification via a mailbox. Default is 16.

**SlavePaths**- Number of slave paths to support. This is the number of sockets and corresponding server threads that will be supported simultaneously for communication with remote master nodes. Default is 8.

#### **Example:**

```
$ SLAVE IPCAL3 40001 1000 10 10
```

The above starts the slave server and names the group global section as "MBTSLVipcal3R". By convention, one typically uses the name of host system or some combination of as the PLC name. The region name parameter passed to the MBP\_MAP\_VPLC would be "ipcal3 (note lower case)". In addition, a server mailbox is created in MBTSLVipcal3MBX. "IPCAL3" would be passed as the Slave Server Mbx Name parameter to the MBP\_REGISTER\_UN SOLICITED call.

#### **2.5 Slave Server Holding Registers**

The Slave Server will support a full complement of 4x holding registers as defined for Modbus PLCs. The administrator of the Host system can specify any portion of the 4x range for use by the Slave Server. The local holding registers can be manipulated by

## **Local Host Slave Server**

### **Direct access to the Virtual PLC (Holding Registers)**

---

other local applications as well as remote Modbus devices, HMIs, and other Host computer systems supporting the Modbus/TCP API.

Data read from or written to the local holding registers is put in proper byte order for the recipient of the data. However, application using the Modbus/TCP API may request that byte ordering be left as is. This can be useful in cases where the information being exchanged is ascii data which has been packed in holding registers. Under normal circumstances, the caller of the API routines will set the FLAG variable to 0 to indicate that the API is responsible for byte ordering. In this case, if a master device writes a  $0001_{16}$  to a Local Slave holding register, it will be read as  $0001_{16}$  if the user has specified a FLAG value of 0. If the user had specified a non-zero value for the FLAG parameter, the API would deliver a value of  $0100_{16}$  to the caller reading the data.

### **2.6 Direct access to the Virtual PLC (Holding Registers)**

The Slave Server holding registers (also referred to the virtual PLC) are contained within a common data structure which is currently a Group Global Region. To map these holding registers in order to read and write to them, use the routine MBP\_MAP\_VPLC. Also use the MBP\_VPLB\_ACCESS locking routine to ensure safe access to the data in the virtual PLC. There is a significant speed advantage in accessing the memory directly. Some users have used the iFIX export and Microsoft Excel macros to define a record map for the holding register global section (see:Exporting Mobus tags from iFIX to a CSV file).

Direct Virtual PLC reads and writes do not go through the slave server, so they will not trigger the unsolicited messaging feature. Keep that in mind when designing your applications.

### **2.7 Unsolicited Messaging**

The Slave Server is capable of forwarding write command messages, from Master devices, directly to a local application. The message is sent to the application encapsulated within a predefined structure named MASTER\_TO\_APPL\_BUF. The original message content from the Master device is preserved within that structure. The definition of the MASTER\_TO\_APPL\_BUF can be found in file MBP\_PEXDEF.H in the MBTDEF\_C.TLB library as shown here. In order for an application to receive unsolicited messages from Master devices on the network, the application must register its interest. Through the registration, the application may designate a specific Master device or group of master devices via the route array parameter. The route array is filled with the information specific to the Master device or devices of interest.

## Local Host Slave Server Unsolicited Messaging

---

### MASTER\_TO\_APPL\_BUF Structure

app\$l_cmd_stat	Frame identification
app\$b_route	5 byte route array
app\$b_msg	<p>Unsolicited message from a master device or system. The Open Modbus/TCP Specification details the contents of the message. Byte order is in the Master's normal byte order.</p> <p>There are structures defined in the file MBP_BUFDEF,H for all of the possible Modbus messages.</p>

+ Frame identification can be any of the following:

- 0x01 = Unsolicited message from a master.
- 0x02 = Confirmation of deregistration request.
- 0x04 = Confirmation of registration request.
- 0x08 = Registration request failed.
- 0x10 = Confirmation of suspend request.
- 0x20 = Confirmation of resume request.

+ Route Array contains the Master's route information so that the receiving application may determine the source of the information.

### **3 MBP Application Library Calls**



# APPLICATION LIBRARY ROUTINES

## MBP\_CRE\_SIGNAL\_OBJ

---

### 3.2 MBP CRE SIGNAL OBJ

Create a signal object used for the asynchronous version calls in the API..

---

**FORMAT** MBP\_CRE\_SIGNAL\_OBJ(WOBJ))

---

**RETURNS** VMS usage: COND\_VALUE  
type: longword  
mechanism: by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS** WOBJ  
type: longword  
access: write  
mechanism: by reference

The address of a longword variable to receive the wait object identifier.

---

**DESCRIPTION** This function returns the identifier for a wait object which is then used in subsequent calls to API routines.

---

**CONDITION VALUES RETURNED** MBP\_SUCCESS Open was successful

SYSTEM SERVICE Condition value returned by an Operating system service call

## APPLICATION LIBRARY ROUTINES

### MBP\_DEL\_SIGNAL\_OBJ

---

#### 3.3 MBP\_DEL\_SIGNAL\_OBJ

Deletes a signal object previously created by a call to MBP\_CRE\_SIGNAL\_OBJ..

---

**FORMAT**            **MBP\_DEL\_SIGNAL\_OBJ(WOBJ)**

---

**RETURNS**            VMS usage:    COND\_VALUE  
                          type:            longword  
                          mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**        WOBJ  
                          type:            longword  
                          access:          write  
                          mechanism:      by reference

The address of a longword variable to receive the wait object identifier.

---

**DESCRIPTION**    This function deletes a synchronization object which was previously created by a call to the function MBP\_CRE\_SIGNAL\_OBJ. All objects will be deleted upon exit of the application if not already deleted by this method.

---

**CONDITION VALUES RETURNED**    MBP\_SUCCESS                    Open was successful

                          SYSTEM SERVICE                Condition value returned by an Operating system service call

## APPLICATION LIBRARY ROUTINES

### MBP\_FORCE\_SINGLE\_COIL(W)

---

#### 3.4 MBP\_FORCE\_SINGLE\_COIL(W)

Force a single output coil.

---

**FORMAT**            **MBP\_FORCE\_SINGLE\_COIL**  
**(MBPPEX,PATH,ROUTE,COIL\_NUMBER,**  
**STATE,STATUS,WOBJ,TMO)**

---

**RETURNS**            VMS usage:    COND\_VALUE  
                          type:            longword  
                          mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below)

---

**ARGUMENTS**        MBPPEX  
                          type:            structure  
                          access:         readonly  
                          mechanism:     by reference

The structure that was returned by the MBP\_OPEN\_NET call.

PATH  
type:            word  
access:         readonly  
mechanism:     by value

Master Path on which the I/O is to be directed. Must be between one and the number of Master Paths allocated by the MBP\_OPEN\_NET call.

ROUTE  
type:            five byte array  
access:         readonly  
mechanism:     by reference

This is an array filled in by the caller that specifies the route path to the particular slave device. The content of this array requires knowledge of the connected network devices and their addresses.

COIL\_NUMBER  
type:            word  
access:         readonly  
mechanism:     by value

The coil number to force in the addressed PLC.

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_FORCE\_SINGLE\_COIL(W)**

---

#### STATE

type: word  
access: readonly  
mechanism: by value

New state of the coil (0 or 1).

#### STATUS

type: structure MBP\_STATUS  
access: write  
mechanism: by reference

Final completion status as returned by the I/O thread handling the operation. A value of MBP\$SUCCESS in the condition field indicates success. See section: 7 for more information.

#### WOBJ

type: long word  
access: read  
mechanism: by value

Synchronization object to be signaled upon completion of the operation. See MBP\_CRE\_SIGNAL\_OBJ for details on creating a synchronization object.

#### TMO

type: long word  
access: read  
mechanism: by value

Timeout value in milliseconds.

---

**DESCRIPTION** This subroutine sets a single coil to a specified state using the given Master path. The wait object is used for I/O synchronization and it must be specified. The object will be signaled when the operation is complete

**APPLICATION LIBRARY ROUTINES**  
**MBP\_FORCE\_SINGLE\_COIL(W)**

---

---

<b>CONDITION VALUES RETURNED</b>	MBP_BADPATH	Path specified not allocated or out of range
	MBP_BADSTARTADDR	Invalid starting address
	MBP_INVALIDEFN	Invalid event flag specified
	MBP_NOTINITIALIZED	MBPPEX structure not initialized
	MBP_PATHINUSE	Path in use
	MBP_SUCCESS	Successful completion
	SYSTEM SERVICE	From SYS\$QIO call.

## APPLICATION LIBRARY ROUTINES

### MBP\_MAP\_VPLC

---

#### 3.5 MBP\_MAP\_VPLC

Map the virtual PLC (also referred to as the slave server holding registers).

---

**FORMAT**            **MBP\_MAP\_VPLC (VPLC\_NAME, VPLC)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**     VPLC\_NAME  
                  type:            null terminated c string  
                  access:          readonly  
                  mechanism:    by reference

The vplc\_name is the name of the slave server to be mapped. The vplc\_name is defined as a parameter given to the slave server when it is started (see: Slave Server Startup).

VPLC  
type:            VPLCBLK structure  
access:          write  
mechanism:      by reference

The structure used to hold information about the virtual PLC for the application. (See VPLCBLK defined in MBP\_PEXDEF in MBTDEF\_C.TLB or MBTDEF\_F.TLB).

---

**DESCRIPTION**    This function maps the virtual PLC region of memory which should have been previously created by the actual slave server application. If it has not already been created, then this routine will return an error that indicates no such slave server region is available on this node. This routine maps the region region which is named by concatenating the string: "MBTSLV", <server name in lower case>, and the letter "R" (IPCAL3 would be: MBTSLVipcal3R). The routine returns access to the virtual PLC registers to the caller in the form of a pointer in the VPLC.regdata\_ptr. Typically, it would point to a known record structure.

---

**CONDITION**    MBP\_MAPVPLCFAIL            Unable to map region  
**VALUES**  
**RETURNED**

**APPLICATION LIBRARY ROUTINES**  
**MBP\_MAP\_VPLC**

---

LIBRARY SERVICE	Condition values returned by LIB\$GET_EF
SYSTEM SERVICE	Condition value returned by SYS\$ENQ

# APPLICATION LIBRARY ROUTINES

## MBP\_OPEN\_NET

---

### 3.6 MBP\_OPEN\_NET

Enable Modbus Communications.

---

**FORMAT**            **MBP\_OPEN\_NET(MBPPEX,DEVICE,MASTER\_PATHS,  
SLAVE\_PATHS)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**     MBPPEX  
                  type:            structure  
                  access:         write  
                  mechanism:    by reference

The structure to receive the starting and ending address of the process specific data area used by the Modbus Plus interface subroutines (see MBPPEX in MBPlus.TLB). This should be a static variable (e.g., by default C creates stack variables and FORTRAN creates static variables).

DEVICE  
type: character  
access: read only  
mechanism: by descriptor

Included only for compatibility with previous Modbus API variations.

MASTER\_PATHS  
type:            word  
access:         read only  
mechanism:     by value

This specifies the number of Master paths the caller wishes to allocate for use. If the entire number of paths specified cannot be allocated, then none are allocated. The total number of paths, both Master and Slave, must not exceed 16.

## APPLICATION LIBRARY ROUTINES

### MBP\_OPEN\_NET

---

SLAVE\_PATHS

type: word  
access: read only  
mechanism: by value

This specifies the number of slave paths to allocate for the purpose of receiving unsolicited messages. See "MBP\_REGISTER\_UN SOLICITED" subroutine entry.

---

#### DESCRIPTION

This subroutine allocates paths to be used for communicating with the Modbus Plus network. The function expands the callers memory space to allocate working storage for future calls to the Modbus/TCP interface. The structure MBPPEX that is returned to the user must be passed on all subsequent calls to the other Modbus/TCP interface routines. This is the first Modbus/TCP function called by the user application. The function also establishes an exit handler. This handler takes care of releasing resources and breaking down connections in cases of unexpected process failure.

The MBPPEX structure is defined in the MBP\_C.TLB text library. The MBPPEX is filled with information necessary to access the process expanded region for subsequent calls to the Modbus/TCP API. The process expanded region is created as a group global temporary memory section. This allows other programs to map the region for analysis purposes while a user program is executing. The name of the region is the string "MBPREG" with the application's process identification number appended to it (e.g. MBPREG00001A2D

The actual master and slave paths allocated are stored in the process-expanded region, returned in the MBPPEX structure, and may be viewed with Modbus Plus process monitor utility.

---

#### CONDITION VALUES RETURNED

MBP_CRMTXFAIL	Unable to create mutex object for path
MBP_MAPFAIL	Unable to expand process region
MBP_PATHFAIL	Unable to allocate a path
MBP_SUCCESS	Open was successful
MBP_THREADCRFAIL	Unable to create a thread to handle I/O on this path
MBP_WRNGNMBRPATHS	Wrong number of paths

## APPLICATION LIBRARY ROUTINES

### MBP\_PRESET\_SINGLE\_REGISTER[W]

---

#### **3.7 MBP\_PRESET\_SINGLE\_REGISTER[W]**

Set a single Programmable Controller register to a specified value.

---

**FORMAT**            **MBP\_PRESET\_SINGLE\_REGISTER[W](MBPPEX,PATH,ROUTE,  
REGISTER\_NUMBER,VALUE,STATUS,WOBJ,TMO)**

---

**RETURNS**            VMS usage:    COND\_VALUE  
                          type:            longword  
                          mechanism:    by value

Longword status as defined by either a system service call or the MBP\_error status codes.

---

**ARGUMENTS**        MBPPEX  
                          type:            structure  
                          access:         readonly  
                          mechanism:     by reference

Structure returned by MBP\_OPEN\_NET.

PATH  
type:            word  
access:         readonly  
mechanism:     by value

Path on which this operation is to be directed. Must be between one and the number of Master paths allocated by the MBP\_OPEN\_NET call.

ROUTE  
type:            five byte array  
access:         readonly  
mechanism:     by reference

This is an array filled in by the caller that specifies the route path to the particular PLC. The content of this array requires knowledge of the connected network devices and their addresses

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_PRESET\_SINGLE\_REGISTER[W]**

---

REGISTER\_NUMBER  
type: longword  
access: readonly  
mechanism: by value

The holding register number to force in the addressed PLC (any valid 4x holding register configured in the target device)

VALUE  
type: word  
access: readonly  
mechanism: by value

New 16 bit value for the register in local Host byte order.

STATUS  
type: structure MBP\_STATUS  
access: write  
mechanism: by reference

Final completion status as returned by the I/O thread handling the operation. A value of MBP\$SUCCESS in the condition field indicates success. See section: 7 for more information.

WOBJ  
type: long word  
access: read  
mechanism: by value

Wait object to signal upon completion of the operation.

TMO  
type: long word  
access: read  
mechanism: by value

Timeout value in milliseconds.

---

**DESCRIPTION** This function sets a single holding register using the specified path. The wait object is used for I/O synchronization. The wait object will be signaled when the operation is complete. The function will insure proper byte order within the PLC addressed by the route array. Both the synchronous and asynchronous versions are available.

## APPLICATION LIBRARY ROUTINES

### MBP\_PRESET\_SINGLE\_REGISTER[W]

---

---

<b>CONDITION VALUES RETURNED</b>	MBP_BADPATH	The specified path is less than or greater than the number of master paths allocated by the MBP_OPEN_NET call.
	MBP_INVALIDDEFN	An invalid wait object value was specified.
	MBP_INVWAITTIME	An invalid wait period was specified. It must be larger than MP\$K_MIN_WAIT.
	MBP_LNKCRFAIL	Unable to allocate a link block for the requested operation.
	MBP_NOTINITIALIZED	MBPPEX returned by MBP_OPEN_NET is corrupt or invalid
	MBP_PATHINUSE	A request to do a second operation on the master path, but the path is already active with a previous request.
	MBP_SUCCESS	Successful completion
	MBP_THREADSIGFAIL	Unable to signal the thread responsible for this path.

## APPLICATION LIBRARY ROUTINES

### MBP\_READ\_EXTENDED[W]

---

#### 3.8 MBP\_READ\_EXTENDED[W]

Read Extended Memory of a Slave Modbus Node.

---

**FORMAT**            **MBP\_READ\_EXTENDED[W](MBPPEX,PATH,ROUTE,START, COUNT,FILE,BUFFER,STATUS,WOBJ,FLAG,TMO)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**     MBPPEX  
                  type:            structure  
                  access:         readonly  
                  mechanism:    by reference

The structure that was returned by the MBP\_OPEN\_NET call.

PATH  
type:            word  
access:         readonly  
mechanism:    by value

Master Path to which this operation is directed. The path must be between one and the number allocated by the MBP\_OPEN\_NET call.

ROUTE  
type:            five byte array  
access:         readonly  
mechanism:    by reference

This is an array filled in by the caller that specifies the route path to the particular PLC. The content of this array requires knowledge of the connected network devices and their addresses.

START  
type:            long  
access:         readonly  
mechanism:    by value

Starting register to read from the addressed slave device. Valid range for each file is 60000 to 69999 register.

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_READ\_EXTENDED[W]**

---

#### COUNT

type: word  
access: readonly  
mechanism: by value

The number of registers to read from the addressed PLC. If the number of registers exceeds what is able to read from a single Modbus transfer, multiple Modbus transfers will be done.

#### FILE

type: word  
access: readonly  
mechanism: by value

The extended memory of the PLC is partitioned into 10000 registers segments, or files, (and the remaining amount for the last segment), allowing registers in the range of 60000 to 69999 per file. The file number ranges from one (1) to the available memory assigned in the PLC for extended memory in ten thousand register increments, to a maximum of 10 for Quantom PLCs and 525 for GE FANAC PLCs.

#### BUFFER

type: array  
access: write  
mechanism: by reference

Caller array to receive the registers from the PLC.

#### STATUS

type: structure MBP\_STATUS  
access: write  
mechanism: by reference

Final completion status as returned by the I/O thread handling the operation. A value of MBP\$SUCCESS in the condition field indicates success. See section: 7 for more information.

#### WOBJ

type: long word  
access: readonly  
mechanism: by value

Wait object to be signaled when I/O is complete.

## APPLICATION LIBRARY ROUTINES

### MBP\_READ\_EXTENDED[W]

---

FLAG  
type: long word  
access: readonly  
mechanism: by value

If non-zero, the data received is not adjusted for byte order.

TMO  
type: long word  
access: read  
mechanism: by value

Timeout value in milliseconds.

---

**DESCRIPTION** This subroutine reads the registers from the extended memory files in the slave PLC on the network. The data is adjusted for normal byte order unless the FLAG byte is set to a non-zero value. The user call may translate into multiple I/Os to the selected target device depending upon the count value. The wait object will be signaled when all of the I/O is complete or an error is encountered.

Note that the starting register number for an extended memory file is 60000, for the other register ranges the first register is x00001. Each extended memory file is partitioned to hold 10000 registers except for the last file which occupies whatever is left in the extended memory. Therefore, a single register file has a potential range from 60000 to 69999. Only a single extended memory file may be written per call.

---

<b>CONDITION VALUES RETURNED</b>	MBP_BADPATH	The Path specified has not been allocated or is out of range
	MBP_BADSTARTADDR	Invalid starting address
	MBP_INVALIDARG	One of the call arguments is invalid, not readable, or not writeable.
	MBP_INVALIDDEFN	Invalid wait object was specified
	MBP_LNKCRFAIL	Unable to create a link block for the new connection
	MBP_MBPPEXINVLD	MBPPEX structure is invalid
	MBP_NOTINITIALIZED	MBPPEX structure not initialized
	MBP_PATHINUSE	The specified path already has an I/O active.

**APPLICATION LIBRARY ROUTINES**  
**MBP\_READ\_EXTENDED[W]**

---

MBP\_SUCCESS

Successful completion

## APPLICATION LIBRARY ROUTINES

### MBP\_READ\_REGISTERS[W]

---

#### 3.9 MBP\_READ\_REGISTERS[W]

Read registers from a Modbus Plus slave node.

---

**FORMAT**            **MBP\_READ\_REGISTERS[W](MBPPEX,PATH,ROUTE,START, COUNT,BUFFER,STATUS,WOBJ,FLAG,TMO)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**     MBPPEX  
                  type:            structure  
                  access:         readonly  
                  mechanism:    by reference

The structure that was returned by the MBP\_OPEN\_NET call.

PATH  
type:            word  
access:         readonly  
mechanism:    by value

Master Path to which the operation is to be directed. The path must be between one and the number allocated by the MBP\_OPEN\_NET call.

ROUTE  
type:            five byte array  
access:         readonly  
mechanism:    by reference

This is an array filled in by the caller that specifies the route path to the particular slave device. The content of this array requires knowledge of the connected network devices and their addresses.

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_READ\_REGISTERS[W]**

---

#### START

type: long  
access: readonly  
mechanism: by value

Starting register in addressed slave device to begin writing user registers to. The most significant digit of a valid address "START" is used to determine if coil status (0xxxx), input status (1xxxx), holding registers (4xxxx), or input registers (3xxxx) are to be read.

#### COUNT

type: word  
access: readonly  
mechanism: by value

The number of registers, or coils, to read from the addressed slave device. If the count is greater than the maximum Modbus transfer size, multiple operations are used to accomplish the full count before return to the caller.

#### BUFFER

type: array  
access: write  
mechanism: by reference

Caller array to receive the register or coil data from the PLC .

#### STATUS

type: structure MBP\_STATUS  
access: write  
mechanism: by reference

Final completion status as returned by the I/O thread handling the operation. A value of MBP\$SUCCESS in the condition field indicates success. See section: 7 for more information.

#### WOBJ

type: long word  
access: readonly  
mechanism: by value

Wait object to be signaled upon I/O completion.

## APPLICATION LIBRARY ROUTINES

### MBP\_READ\_REGISTERS[W]

---

FLAG  
type: long word  
access: readonly  
mechanism: by value

If FLAG is non-zero, the data is not byte swapped when it is placed into the caller's buffer (applies only to registers, not coils).

TMO  
type: long word  
access: read  
mechanism: by value

Timeout value in milliseconds. Must be greater than MP\$K\_MIN\_WAIT.

---

**DESCRIPTION** This function reads the specified registers or coils from the target slave device. The wait object is used for synchronization. The wait object is signaled when all of the registers, or coils, requested have been read. To satisfy the request, this routine may do more than a single Modbus transfer. The size of the buffer should be sized for either 8 bits per byte for discretetes (0xxxx, 1xxxx), or two bytes per register (3xxxxx, 4xxxxx). All reads, of registers, are put in normal Host byte order, unless the FLAG argument is set to a non-zero value.

This subroutine is available in the synchronous and asynchronous versions.

---

<b>CONDITION VALUES RETURNED</b>	MBP_BADPATH	Path specified not allocated or out of range
	MBP_BADSTARTADDR	Invalid starting address
	MBP_INVALIDARG	One of the call arguments is invalid, not readable, or not writeable.
	MBP_INVALIDDEFN	Invalid event flag specified
	MPB_INVLPATH	Specified path is not between one and the number allocated by MBP_OPEN_NET
	MBP_MBPPEXINVLD	MBPPEX structure Is invalid
	MBP_NOTINITIALIZED	MBPPEX structure not initialized

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_READ\_REGISTERS[W]**

---

MBP_PATHINUSE	The specified path already has an operation active.
MBP_SUCCESS	Successful completion
MBP_THREADSIGFAIL	Unable to signal the I/O thread for the specified path.

## APPLICATION LIBRARY ROUTINES

### MBP\_READ\_UNSOLICITED

---

#### 3.10 MBP\_READ\_UNSOLICITED

Read unsolicited mail from a Master Device or System Unsolicited mail is the result of a Master device writing to the local Slave Server holding registers. A copy of the actual Master message is forwarded to all applications who have registered to receive unsolicited messages from the Master.

---

**FORMAT**            **MBP\_READ\_UNSOLICITED(MBPPEX,BUFFER,PATH,WOBJ,SIZE,REQ\_SIZE)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MBP\_error status codes.

---

**ARGUMENTS**     MBPPEX  
                  type:            structure  
                  access:        readonly  
                  mechanism:    by reference

Structure returned by MBP\_OPEN\_NET.

**BUFFER**  
type:            array  
access:        write  
mechanism:    by reference

Caller array to receive the unsolicited message data. The message is in the form of structure MASTER\_TO\_APPL\_BUF.

**PATH**  
type:            longword  
access:        readonly  
mechanism:    by value

The Slave Path to which the read is to be associated.

## APPLICATION LIBRARY ROUTINES

### MBP\_READ\_UNSOLICITED

---

WOBJ  
type: longword  
access: read  
mechanism: by value

Object to be signaled upon reception of a message.

SIZE  
type: longword  
access: write  
mechanism: by reference

The size in bytes of the entire message. The message is encapsulated in a structure defined as MASTER\_TO\_APPL\_BUF defined in header file MBP\_PEXDEF.H. The actual message data is prefixed by an identifier longword and the route array defining type and the source of the message. See the section pertaining to the Slave Server for a description of the Unsolicited message concept.

REQ\_SIZE  
type: longword  
access: write  
mechanism: by value

User specified size for the expected message. If the received message is larger than this value, a status of SS\_BUFFEROVF is returned to indicate that the message was truncated.

---

#### **DESCRIPTION**

This function will read unsolicited messages received via the designated Slave Path mailbox. The mailbox was created by a previous call to the MBP\_REGISTER\_UNSOLICITED function..

The routine is passed the Slave Path on which to post the read. If there is no message currently available on the designated path, this function waits. The routine places a read on the mailbox for the path. Asynchronous I/O is possible, depending upon operating system, by using standard I/O calls. On OpenVMS the programmer may use the QIO system service to accomplish this. The channel, which is returned by the MBP\_REGISTER\_UNSOLICITED function, is used for the I/O call.

The message received is a structure named MASTER\_TO\_APPL\_BUF as defined in the Modbus header file mbp\_bufdef.h. The first longword in the structure indicates the type of message contained in the structure. The second field in the structure is the route array from the source device. No byte swapping is done on the data. If the data is true sixteen bit registers, the bytes may need to be swapped to convert them to the OpenVMS sixteen bit word format. If the data contains ASCII data, or is a group of discretes packed 8 to a byte, then do not swap the bytes.

## APPLICATION LIBRARY ROUTINES

### MBP\_READ\_UNSOLICITED

---

---

<b>CONDITION VALUES RETURNED</b>	MBP_NOTINITIALIZED	MBPPEX returned by MBP_OPEN_NET is corrupt or invalid.
	MBP_BADPATH	The specified path is less than or greater than the number of slave paths allocated by the MBP_OPEN_NET call.
	MBP_PATHINUSE	A request to do a second read or write to the master path, but the path is already active with a previous I/O request.
	SS\$_BUFFEROVF	User buffer was not large enough to contain the complete message.
	SS\$_MBTOOSML	The user has requested a buffer larger than the size of mailbox. The user buffer should not be larger than MP\$K_UNSOLICITED.
	SYSTEM	Result of VMS SYS\$QIO call

## APPLICATION LIBRARY ROUTINES

### MBP\_REGISTER\_UNSOLICITED

---

#### 3.11 MBP REGISTER UNSOLICITED

Establish a mailbox for a Slave Path, to which the local Slave Server may deliver unsolicited messages. The unsolicited messages are the result of a Master device writing to the local holding registers.

---

**FORMAT**            **MBP\_REGISTER\_UNSOLICITED (MBPPEX,MBX\_NAME, SLAVE\_SERVER\_MBX\_NAME,MSG\_CNT,PERM,MBX\_CHANNEL, SLAVE\_PATH,ROUTE,WOBJ)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below)

---

**ARGUMENTS**     MBPPEX  
                  type:            structure  
                  access:          write  
                  mechanism:    by reference

The structure returned by a previous call to MBP\_OPEN\_NET.

**MBX\_NAME**  
type:            null terminated character string  
access:          read only  
mechanism:      by reference

Specifies the name of the mailbox to be created for delivery of messages to the specified Slave Path.

**SLAVE\_SERVER\_MBX\_NAME**  
type:            null terminated character string  
access:          read only  
mechanism:      by reference

Specifies the name of the mailbox used by the Slave Server of choice. This should contain the server mailbox name (see section: Slave Server Startup).

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_REGISTER\_UNSOLICITED**

---

MSG\_CNT

type: word  
access: read only  
mechanism: by value

Specifies the maximum number of messages that are allowed to be queued in the Slave Path mailbox specified by MBX\_NAME. The maximum value for this parameter will be a function of your account quotas. The number of messages specified will be used in conjunction with the size of a MASTER\_TO\_APPL\_BUF structure to determine the total buffer space required to support the mailbox.

PERM

type: word  
access: read only  
mechanism: by value

A value of 1 specifies that the mailbox should be established as a permanent mailbox. A value of 0 indicates a temporary mailbox. A permanent mailbox will allow messages to be retained over process activation, but not over system bootstraps. Temporary mailboxes are typically used unless there is a specific need to preserve messages across application restarts.

MBX\_CHANNEL

type: long word  
access: write  
mechanism: by reference

The actual channel number assigned to the mailbox is returned to the caller. This channel should be used for asynchronous access to messages received via the mailbox. System services supplied by the operating system can be used to accomplish this. On OpenVMS systems, the caller may use the SYS\$QIO system service with the returned channel value.

SLAVE\_PATH

type: word  
access: readonly  
mechanism: by value

The Slave Path to associate with the mailbox and the registration route.

## APPLICATION LIBRARY ROUTINES

### MBP\_REGISTER\_UNSOLICITED

---

#### ROUTE

type: character array  
access: readonly  
mechanism: by reference

The route information for the Master Device or Devices of interest. As explained in the description above, the caller may supply a standard route specification for a single Master Device, or use wildcard values of 255<sub>10</sub> for any or all of the five bytes of route information.

#### WOBJ

type: longword  
access: readonly  
mechanism: by value

A wait object created by a call to the MBP\_CRE\_SIGNAL\_OBJ function. This function will wait for the object to be signaled upon completion of the operation.

---

#### DESCRIPTION

Registration for unsolicited messages allows the caller to specify an interest in a specific Master device or group of Master devices. This is accomplished by supplying the route information for the Master device or devices. The five byte route array is filled with the designation for the Master device. A group of Master devices may be specified by using a wildcard value in any or all of the bytes of the route array. The wildcard value is 255<sub>10</sub>. For example, if the caller was interested in a single Master device, whose route array was 10.22.33.44.1, you would supply those five bytes in the route array. If the caller was interested in all Masters in domain 10.22.33, you could supply 10.23.33.255.255 in the route array. A write to the local holding registers from any Master fitting this filter would result in an unsolicited message delivery to the Slave path's mailbox.

If the mailbox is to be created as a permanent mailbox, the caller must have "SYSNAM" privilege. The reading process can be either the current process or any other process in the system, providing it is aware of the mailbox. The **MBP\_READ\_UNSOLICITED function may be called to actually read messages from the mailbox.**

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_REGISTER\_UNSOLICITED**

---

<b>CONDITION VALUES RETURNED</b>		
	MBP_INVALIDEFN	An invalid wait object value was supplied for synchronization.
	MBP_MBXCREFAIL	Creation of the local mailbox failed. Most likely causes for this failure are quota and privilege related issues. Permanent mailboxes require SYSNAM privilege.
	MBP_MBXFAIL	Unable to create mailbox (see secondary status, will contain the system service error code)
	MBP_NAMEFAIL	Unable to translate the name associated with the local mailbox.
	MBP_NOSLAVE	No slave paths were allocated
	MBP_NOTINITIALIZED	MBPPEX structure not initialized
	MBP_SLVASSIGNED	The slave path is already in use.
	MBP_SLVBUSY	Attempt to allocate second read on same slave path
	MBP_SLVREADFAIL	Slave read failure
	MBP_SUCCESS	Successful completion

## APPLICATION LIBRARY ROUTINES

### MBP\_RESUME\_UNSOLICITED

---

#### 3.12 MBP\_RESUME\_UNSOLICITED

Resume the transmission of unsolicited messages to the callers specified Slave Path mailbox, which had previously been suspended by a call to MBP\_SUSPEND\_UNSOLICITED.

---

**FORMAT**            **MBP\_RESUME\_UNSOLICITED (MBPPEX,SLAVE\_PATH,ROUTE,WOBJ)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**     MBPPEX  
                  type:            structure  
                  access:         read  
                  mechanism:    by reference

The structure returned by a previous call to MBP\_OPEN\_NET.

SLAVE\_PATH  
type:            word  
access:         readonly  
mechanism:     by value

Slave Path on which to resume unsolicited message reception.

ROUTE  
type:            character array  
access:         readonly  
mechanism:     by reference

A five byte array specifying the same route as used in the registration request on this path. It must match the original specification exactly.

## APPLICATION LIBRARY ROUTINES

### MBP\_RESUME\_UNSOLICITED

---

WOBJ  
type: longword  
access: readonly  
mechanism: by value

A signal object upon which to wait for completion of the operation requested. The object must have been created by a previous call to MBP\_CRE\_SIGNAL\_OBJ..

---

**DESCRIPTION** This function allows the caller to resume transmission of unsolicited messages from the Slave Server to the designated Slave Path mailbox. The caller must have suspended unsolicited messaging via a call to MBP\_SUSPEND\_UNSOLICITED previously. A confirmation message is sent by the Slave Server to the Slave Path mailbox as an acknowledgment of the call to resume.

---

<b>CONDITION VALUES RETURNED</b>	MBP_FAILURE	The route specified did not match the route information registered for this path.
	MBP_MBXFAIL	Unable to create mailbox (see secondary status, will contain the VMS system service code
	MBP_NOREGUNSOL	Not registered for unsolicited messages on this Slave Path.
	MBP_NOSLAVE	No Slave Paths were allocated. )
	MBP_NOSUCHPATH	The desired slave path was not allocated to the target process.
	MBP_NOTSUSPND	Unsolicited reception of messages was not suspended on this path.
	MBP_SUCCESS	Successful resumption of unsolicited message reception for this path.

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_SUSPEND\_UNSOLICITED**

---

#### **3.13 MBP SUSPEND UNSOLICITED**

Suspends the transmission of Modbus messages to the callers mailbox..

---

**FORMAT**            **MBP\_SUSPEND\_UNSOLICITED**  
**(MBPPEX,PROCESS\_NAME,DEVICE,CTRL\_SLAVE\_PATH)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**    MBPPEX  
                  type:            structure  
                  access:          read  
                  mechanism:    by reference

The structure returned by a previous call to MBP\_OPEN\_NET.

SLAVE\_PATH  
type:            word  
access:          readonly  
mechanism:      by value

Slave Path on which to suspend unsolicited message reception.

ROUTE  
type:            character array  
access:          readonly  
mechanism:      by reference

A five-byte array specifying the same route as used in the registration request on this path. It must match the original specification exactly.

WOBJ  
type:            longword  
access:          readonly  
mechanism:      by value

A signal object upon which to wait for completion of the operation requested. The object must have been created by a previous call to MBP\_CRE\_SIGNAL\_OBJ.

---

## APPLICATION LIBRARY ROUTINES

### MBP\_SUSPEND\_UNSOLICITED

---

**DESCRIPTION** This function allows the caller to suspend the reception of unsolicited Modbus messages. The caller must have previously registered for unsolicited message reception. The caller must provide the same route array contents which were present in the call to register on this path.

A suspend request message is sent to the Slave Server associated with this path. The Slave Server checks the request and suspends its transmissions to the application. An acknowledge message is queued by the Slave Server, to the callers local mailbox to confirm the request.

---

<b>CONDITION VALUES RETURNED</b>	MBP_ALRSUSPND	Unsolicited messaging is already suspended on this Slave Path.
	MBP_FAILURE	The operation failed. The secondary status value can be found in the MBPPEX structure.
	MBP_INVALIDEFN	An invalid wait object value was supplied.
	MBP_NOSLAVE	No Slave Paths were allocated)
	MBP_NOSUCHPATH	The desired Slave Path was not allocated to the target process.
	MBP_NOINITIALIZED	The MBPPEX structure is invalid. MBP_OPEN_NET must be called prior to calling this function.
	MBP_NOREGUNSOL	The calling application has not registered for unsolicited messaging on this path.
	MBP_SUCCESS	Successful completion of the request.

## APPLICATION LIBRARY ROUTINES

### MBP\_VPLC\_ACCESS

---

#### 3.14 MBP VPLC ACCESS

Change the application's lock mode for the virtual PLC.

---

**FORMAT**            **MBP\_VPLC\_ACCESS (VPLC, MODE)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**     VPLC  
                  type:            VPLCBLK structure  
                  access:          modify  
                  mechanism:    by reference

The structure used to hold information about the virtual PLC for the application. (See MBP\_PEXDEF in MBTDEF\_C.TLB or MBTDEF\_F.TLB).

MODE  
type:            longword  
access:          readonly  
mechanism:      by value

Type of access requested. Can be:

CON\$K\_NULL – Null mode  
CON\$K\_READER – Concurrent read mode  
CON\$K\_WRITER – Exclusive write mode

---

**DESCRIPTION**    This function is called in order to control access to the virtual PLC (slave server register region). Your application should use a different mode depending on whether it is doing a read or a write. Writers should request exclusive mode (CON\$K\_WRITER). Readers should request shared mode (CON\$K\_READER). The event flag allocated by LIB\$GET\_EF during the MBP\_MAP\_VPLC call is used when converting the lock.

**It is extremely important** that the application release hold on the lock by requesting CON\$K\_NULL mode when finished reading or writing. If not, this will cause the slave server application to be blocked from updating and possibly even reading the virtual PLC.

---

## APPLICATION LIBRARY ROUTINES

### MBP\_VPLC\_ACCESS

---

<b>CONDITION VALUES RETURNED</b>	MBP_INVALIDARG	An invalid mode argument was specified.
	MBP_SUCCESS	Successful completion of the operation
	SYSTEM SERVICE	Condition values returned by the SYS\$ENQ service.

## APPLICATION LIBRARY ROUTINES

### MBP\_WRITE\_EXTENDED[W]

---

#### **3.15 MBP WRITE EXTENDED[W]**

Write extended memory files in a Modbus Slave node.

---

<b>FORMAT</b>	<b>MBP_WRITE_EXTENDED[W](MBPPEX,PATH,ROUTE, START,COUNT,FILE,BUFFER,STATUS,WOBJ,FLAG[,TMO])</b>
---------------	-----------------------------------------------------------------------------------------------------

---

<b>RETURNS</b>	VMS usage: COND_VALUE type: longword mechanism: by value
----------------	----------------------------------------------------------------

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

<b>ARGUMENTS</b>	MBPPEX type: structure access: readonly mechanism: by reference
------------------	--------------------------------------------------------------------------

The structure that was returned by the MBP\_OPEN\_NET call.

PATH  
type: word  
access: readonly  
mechanism: by value

Master Path on which to execute the operation. The path must be between one and the number of Master Paths allocated by the MBP\_OPEN\_NET call.

ROUTE  
type: five byte array  
access: readonly  
mechanism: by reference

This is an array filled in by the caller that specifies the route path to the particular device. The content of this array requires knowledge of the connected network devices and their addresses.

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_WRITE\_EXTENDED[W]**

---

#### START

type: long  
access: readonly  
mechanism: by value

Starting register in addressed PLC to beginning reading registers from. Valid range for each register file is 60000 to 69999.

#### COUNT

type: word  
access: readonly  
mechanism: by value

The number of registers to read from the addressed device. If the number of registers exceeds what is able to read from a single Modbus transfer, multiple Modbus transfers will be performed to complete the operation.

#### FILE

type: word  
access: readonly  
mechanism: by value

The extended memory of the PLC is partitioned into 10000 registers segments, or files, (and the remaining amount for the last segment), allowing registers in the range of 60000 to 69999 per file. The file number ranges from one (1) to the available memory assigned in the PLC for extended memory in ten thousand register increments, to a maximum of 10 for Quantom PLCs and 525 for GE FANAC PLCs.

#### BUFFER

type: array  
access: readonly  
mechanism: by reference

Caller array of registers to be written to the PLC register file.

#### STATUS

type: structure MBP\_STATUS  
access: write  
mechanism: by reference

Final completion status as returned by the I/O thread handling the operation. A value of MBP\$SUCCESS in the condition field indicates success. See section: 7 for more information.

## APPLICATION LIBRARY ROUTINES

### MBP\_WRITE\_EXTENDED[W]

---

WOBJ  
type: long word  
access: readonly  
mechanism: by value

Object to be signaled upon completion of the operation. The object must have been previously allocated by calling the MBP\_CRE\_SIGNAL\_OBJ.

FLAG  
type: long word  
access: readonly  
mechanism: by value

A value of 0 indicates that the data is to be put in proper byte order for the destination. A non-zero value indicates that the caller wishes to control byte order.

TMO  
type: long word  
access: read  
mechanism: by value

A timeout value in milliseconds to allow before a timeout condition is signaled.

---

**DESCRIPTION** This function writes registers in extended memory files in a slave device on the network. Data is adjusted for proper byte order if the FLAG parameter is 0. If the FLAG argument is non-zero it is assumed that the caller will control byte order. The call is available in synchronous and asynchronous forms. The WOBJ object is signaled upon completion of the operation.

Note that the starting register number for an extended memory file is 60000, for the other register ranges the first register is x00001. Each extended memory file is partitioned to hold 10000 registers except for the last file which occupies whatever is left in the extended memory. Therefore, a single register file has a potential range from 60000 to 69999. Only a single extended memory file may be written per call.

---

<b>CONDITION VALUES RETURNED</b>	MBP_BADMTXOBJ	An internal mutex object was invalid.
	MBP_BADPATH	Path specified not allocated or out of range
	MBP_BADSTARTADDR	Invalid address

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_WRITE\_EXTENDED[W]**

---

MBP_INVALIDDEFN	Invalid event flag specified
MBP_INVALIDROUTE	An invalid route value was supplied. All bytes must be non-zero.
MPB_INVLPATH	Specified path is not between one and the number allocated by MBP_OPEN_NET
MBP_LNKCRFAIL	Unable to create a link block to use on this operation.
MBP_PATHINUSE	The specified path already has an I/O active.
MBP_NOLINKSAVL	Maximum number of links are active on this path already.
MBP_NOTINITIALIZED	MBPPEX structure not initialized
MBP_PATHINUSE	The designates Master Path already has an operation in progress.
MBP_SUCCESS	Successful completion of the operation

## APPLICATION LIBRARY ROUTINES

### MBP\_WRITE\_REGISTERS

---

#### 3.16 MBP WRITE REGISTERS

Write registers to a Modbus slave device.

---

**FORMAT**            **MBP\_WRITE\_REGISTERS[W](MBPPEX,PATH,ROUTE,  
START,COUNT,BUFFER,STATUS,WOBJ,FLAG,TMO)**

---

**RETURNS**        VMS usage:    COND\_VALUE  
                  type:            longword  
                  mechanism:    by value

Longword status as defined by either a system service call or the MPB\_xxxx status codes (see error codes listed below).

---

**ARGUMENTS**     MBPPEX  
                  type:            structure  
                  access:          readonly  
                  mechanism:    by reference

The structure that was returned by the MBP\_OPEN\_NET call.

PATH  
type:            word  
access:          readonly  
mechanism:      by value

Master Path on which to execute the operation. The path must be between one and the number of Master Paths allocated by the MBP\_OPEN\_NET call.

ROUTE  
type:            five byte array  
access:          readonly  
mechanism:      by reference

This is an array filled in by the caller that specifies the route path to the particular Slave device. The content of this array requires knowledge of the connected network devices and their addresses.

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_WRITE\_REGISTERS**

---

#### START

type: long  
access: readonly  
mechanism: by value

Starting register in addressed Slave device to begin writing user data. The START register is used to determine whether registers or coils are being acted upon. Coils are being addressed in the 0x and 1x ranges and registers are addressed in the 3x and 4x ranges.

#### COUNT

type: word  
access: readonly  
mechanism: by value

The number of registers or coils to write to the addressed PLC. If the number of registers exceeds what is able to be written in a single Modbus transfer, multiple Modbus transfers will be used.

#### BUFFER

type: array  
access: read  
mechanism: by reference

Caller array containing the new coil or register data to be written to the Slave device.

#### STATUS

type: structure MBP\_STATUS  
access: write  
mechanism: by reference

Final completion status as returned by the I/O thread handling the operation. A value of MBP\$SUCCESS in the condition field indicates success. See section: 7 for more information.

#### WOBJ

type: long word  
access: readonly  
mechanism: by value

Object to be signaled upon completion of the operation.

## APPLICATION LIBRARY ROUTINES

### MBP\_WRITE\_REGISTERS

---

FLAG  
type: long word  
access: readonly  
mechanism: by value

A value of 0 indicates that proper byte order is maintained for the caller. A non-zero value indicates the caller will control byte order (applies only to registers, not coils).

TMO  
type: long word  
access: read  
mechanism: by value

A timeout value in milliseconds to allow before a timeout condition is signaled.

---

**DESCRIPTION** This subroutine writes the user buffer to the registers or coils within the Slave device addressed by the specified route array.

The wait object is signaled upon completion of the operation. The object must have been allocated by calling the MBP\_CRE\_SIGNAL\_OBJ function

The START register number is used to determine which type of output is being performed, register or coil. The source buffer should be sized for two bytes per holding register, or eight coils per byte for coils.

A zero FLAG parameter indicates correct byte order is to be maintained. A non-zero value indicates that the caller will maintain byte order.

Both the synchronous and asynchronous version is available.

---

<b>CONDITION VALUES RETURNED</b>	MBP_MADMTXOBJ	An internal mutex object was invalid.
	MBP_BADPATH	Path specified not allocated or out of range
	MBP_BADSTARTADDR	Invalid starting address
	MBP_INVALIDARG	One of the call arguments is invalid, not readable, or not writeable.
	MBP_INVALIDDEFN	Invalid event flag specified

## **APPLICATION LIBRARY ROUTINES**

### **MBP\_WRITE\_REGISTERS**

---

MPB_INVLPATH	Specified path is not between one and the number allocated by MBP_OPEN_NET
MPB_INVALIDROUTE	An invalid route value was supplied. Must be non-zero.
MPB_LNKCRFAIL	Unable to allocate a link block.
MPB_NOLINKSAVL	Maximum links already in use on this path.
MPB_NOTINITIALIZED	MBPPEX structure not initialized
MPB_PATHINUSE	The specified path already has an I/O active.
MPB_SUCCESS	Successful completion of the operation.
MPB_THREADSIGFAIL	Signaling of the I/O thread for this path has failed.

## **4 Example Programs**

## **TEST PROGRAMS**

### **Introduction Test Programs**

---

#### **4.1 Introduction Test Programs**

The following test utilities will help the programmer evaluate the operation of the API library once installed on the host system.

- 1) TEST\_OPEN\_CLOSE.C – tests the ability of the API library to create the data structures and threads required to carry out all other MBTCP calls.
- 2) TEST\_READS.C – Tests the ability of the API to perform a read from a set of 4x registers on a specified slave device.
- 3) TEST\_WRITES.C – Tests the ability of the API calls to write information to a set of 4x registers on a specified slave device.
- 4) TEST\_READS.FOR – Same as above except written in FORTRAN.
- 5) TEST\_READ\_WRITE.C – Tests the operation of the singular read/write operation. Both a read and write operation are performed in the target slave device. **\*\*NOTE\*\*** See the specific Modicon Device manual for information about how the read/write operation is implemented for the device. The order of the write and read operations vary from device to device. Results of overlapped read/write areas are undefined.
- 6) TEST\_WRITER\_UN SOLICITED – Is basically the same as TEST\_WRITES. It was used in writing to a local MBT\_SLAVE\_SERVER process for testing of unsolicited messaging.
- 7) UNSOLICITED\_LISTENER – Is an application used for testing unsolicited messaging from an MBT\_SLAVE\_SERVER application on the local host system.
- 8) LTEST – A FORTRAN application that tests writing to the virtual PLC using the library access routines. It writes values, reads them back and dumps them to the screen.
- 9) LTEST\_MAP – A FORTRAN program that does exactly the same thing as LTEST does, except it uses the MBP\_MAP\_VPLC and MBP\_VPLC\_ACCESS routines. The reads and writes are done without using the MBP access routines. There is a significant speed advantage in bypassing the slave server for reads and writes.
- 10) TEST\_RW\_VPLC – A C application that writes some values to the virtual PLC, reads them back and dumps the values. This application uses direct mapping, MBP\_MAP\_VPLC.
- 11) TEST\_MSTR – Is a fairly rigorous test of the MBTCP package as a whole. It requires that an MBT\_SLAVE\_SERVER application be running on the local host system. It also requires that an MSTR Instruction be prepared on a target slave PLC. The application then uses user input values to formulate a set of data for the MSTR instruction to use in accessing the local MBT\_SLAVE\_SERVER holding registers. Once the application has written the information required by the MSTR instruction, it triggers the MSTR by activating the MSTR control coil. The MSTR then writes data to the local MBT\_SLAVE\_SERVER holding registers. This information is then verified by the TEST\_MSTR application.



## **5 UTILITIES**

# UTILITIES

## READREG DCL Utility

### 5.1 READREG DCL Utility

The ReadReg utility is responsible for displaying data to the user or placing the data into DCL symbols. It is invoked as a foreign command. Optional switches are enclosed in braces. The following assignment must be made to use the utility:

```
$ rr == "$dcl_readreg
```

#### 5.1.1 ReadReg Command Syntax

**\$ rr /reg= Starting holding register**

```
[/route=(xx.xx.xx.xx.xx)]  
[/local=<region_name>]  
/count=<>  
[/byte]  
[/hex]  
[/ascii]  
[/force]  
[/fmt=<format-string>]  
[/symbols]  
[/regfile=n]
```

##### **Switch Definition:**

**/reg=** Starting holding or extended starting register for read. This may be omitted if the DCL symbol: MBP\_START\_REGISTER populated with a valid holding or extended register. The value should be between 40000 to 49999 or 400000 to 4099999 for holding registers and 60000 to 69999 or 600000 to 6099999 for extended registers.

**/regfile=n** If present, represents the extended memory register file. This also effects the meaning of the “/reg” parameter.

**/route=** Route path to the PLC. This may be omitted if the DCL symbol: MBP\_ROUTE populated with a valid route path. This is the IP address plus the unit of the Modbus node on the Ethernet. It should be specified in normal dot format (e.g. 10.0.0.1.1) where 10.0.0.1 is the IP address and the last one is the unit.

**/local=<region-name>** If present, then map the virtual plc section and access the data directly instead of using the access routines which request the data from the virtual server. Keep in mind the case sensitivity. If the region name is lowercase, you must use quotes (“”) around the name.

**/byte=** Byte swap the data before displaying

**/hex=** Display data in hex

**/ascii=** Display or store data in symbol as ASCII. First null in holding register read will terminate string.

## UTILITIES

### READREG DCL Utility

---

**/fmt=<format-string>** = Format the raw data into displayable ASCII data fields. The following formats statements are valid:

**<repeat>** - Indicates the number of times this format should be repeated. For the input data, before proceeding to the next input value.

**<size>** - Indicates the number of bytes to use in the formatted output string. If too few bytes given, it will be increased automatically to allow for display of the data value. For C data types, it indicates the number of bytes to transfer from the input to output data buffer.

**<repeat>C<size>** - Character

**<repeat>B<size>** - Byte integer

**<repeat>W<size>** - Word Integer

**<repeat>I<size>** - Longword integer

**<repeat>D<size>** - Quadword integer

**<repeat>F<size>** - Floating Point

(Assumes stored as IEEE 2 16 bit registers byte swapped if necessary)

**<repeat>X<size>** - Skip Data bytes in the input buffer

**/symbols=** If present, then this utility will populate DCL symbols to HR\_1 to HR\_nn where nn= count if "/ASCII" is not specified. Otherwise, HR\_ASCII is set to the ASCII text.

# UTILITIES

## READREG DCL Utility

---

### 5.1.2 ReadReg Examples

```
#!  
#! IPACT Quantum test is at 10.0.3.2  
#! PLC Clock register in PLC is at  
#! 406500  
#!  
$ rr /reg=406500-  
  /route=10.0.3.2.1-  
  /count=8-  
  /symbols  
$ show sym HR_*
```

Parsing: RDREG /REG=406500 /ROUTE=10.0.3.2.1 /COUNT=8 /SYMBOLS

Route path: 010.000.003.002.001

```
406500 | 96  
406501 | 1024  
406502 | 768  
406503 | 7936  
406504 | 1024  
406505 | 2560  
406506 | 2048  
406507 | 15104  
HR_000 = " 96"  
HR_001 = " 1024"  
HR_002 = " 768"  
HR_003 = " 7936"  
HR_004 = " 1024"  
HR_005 = " 2560"  
HR_006 = " 2048"  
HR_007 = "15104"
```

```
$ rr /route=10.0.1.3 /reg=40001 /fmt="25B10" /count=40 /byte
```

Parsing: RDREG /ROUTE=10.0.1.3 /REG=40001 /FMT="25B10" /COUNT=40 /BYTE

Route path: 010.000.001.003.000

```
101, 0, 102, 0, 103, 0, 104, 0, 105, 0,  
106, 0,  
107, 0, 108, 0, 109, 0, 110, 0, 0, 0,  
0, 0,  
0
```

```
#!  
#! Floating point read from file 10  
#!  
$ rdreg/reg=60000-  
  /route=10.0.3.2.1-  
  /count=2-  
  /fmt="f10.0"-  
  /regfile=10/byte
```

## **UTILITIES**

### **READREG DCL Utility**

---

Parsing: RDREG /REG=60000 /ROUTE=10.0.3.2.1 /COUNT=2 /FMT="f10.0"  
/REGFILE=10/BY  
TE  
Reading Extended Registers, Register file: 10  
Route path: 010.000.003.002.001  
Format string: f10.0  
10010.000000

# UTILITIES

## WRITEREG DCL Utility

---

### 5.2 WRITEREG DCL Utility

The WriteReg utility may be used to write data to the PLC. It is invoked as a foreign command. Optional switches are enclosed in braces. The following assignment must be made to use the utility:

```
$ wr == "$dcl_writereg
```

#### 5.2.1 WriteReg Command Syntax

```
$ wr /reg= Starting holding register  
  [/route=(xx.xx.xx.xx.xx)]  
  [/local=<region_name>]  
  [/byte]  
  /fmt=<format-string>  
  /data=<format-string>  
  [/regfile=n]
```

##### Switch Definition:

**/reg=** Starting holding or extended starting register for write. This may be omitted if the DCL symbol: MBP\_START\_REGISTER populated with a valid holding or extended register. The value should be between 40000 to 49999 or 400000 to 4099999 for holding registers and 60000 to 69999 or 600000 to 609999 for extended registers.

**/regfile=n** If present, represents the extended memory register file. This also effects the meaning of the “/reg” parameter.

**/route=** Route path to the PLC. This may be omitted if the DCL symbol: MBP\_ROUTE populated with a valid route path. This is the IP address plus the unit of the Modbus node on the Ethernet. It should be specified in normal dot format (e.g. 10.0.0.1.1) where 10.0.0.1 is the IP address and the last one is the unit.

**/local=<region-name>** If present, then map the virtual plc section and access the data directly instead of using the access routines which request the data from the virtual server. Keep in mind the case sensitivity. If the region name is lowercase, you must use quotes (“”) around the name.

**/byte=** Byte swap the data before displaying

**/fmt=<format-string>** = Format the data is be stored as in the PLC. The following formats statements are valid:

```
<repeat>C<size> - Character  
<repeat>B - Byte integer  
<repeat>W - Word Integer  
<repeat>I - Longword integer  
<repeat>D - Quadword integer  
<repeat>F - Floating Point
```

## UTILITIES

### WRITEREG DCL Utility

---

(Assumes stored as IEEE in the Virtual PLC)

<repeat> - Indicates the number of times this format should be repeated. For the input data, before proceeding to the next input value.

<size> - Indicates the number of bytes to use in writing the character data.

**/data=** comma separated string of data

#### **5.2.2 WriteReg Example**

```
$ wr /reg=40001/route="10.0.1.3"/fmt="I,D,c30"/data="-  
2147483648,214748364712345,How were you Yesterday"  
Parsing: WRREG /REG=40001/ROUTE="10.0.1.3"/FMT="I,D,c30"/DATA="-  
2147483648,214748364712345,How were you Yesterday"  
Route path: 010.000.001.003.000
```

## **6 Header Files**

## **Header Files**

### **“C” Header Files**

#### **6.1 “C” Header Files**

The following structures are provided in the text library MBPDEF\_C.TLB:

- 1) MBP\_PEXDEF.H- Definition of all structures used by the API library
- 2) MBP\_BUFDEF.H- Definition of all message structures used by the Modbus/TCP protocol.
- 3) MBPPEX.H- Definition of the MBPEX structure used by all API calls.
- 4) MBPSHARE\_MSG.H- Definition of all MBP error codes in “C” format.
- 5) MBPDEF.H- Prototypes for all the API function calls

#### **6.2 Header Files**

The following structures are provided in the text library MBPDEF\_F.TLB:

- 1) MBP\_PEXDEF.TXT- Definition of all structures used by the API library
- 2) MBP\_BUFDEF.TXT- Definition of all message structures used by the Modbus/TCP protocol.
- 3) MBPPEX.TXT- Definition of the MBPEX structure used by all API calls.
- 4) MBPSHARE\_MSG.TXT- Definition of all MBP error codes in “FORTRAN” format.
- 5) MBPDEF.TXT- Definitions for all the API function calls.

## **7 MBP Error Codes**

## **MBP Error Codes**

### **API Error Status Returns**

#### **7.1 API Error Status Returns**

The MBTCP software errors are listed below along with the recovery procedures. Additional errors may be returned by the OpenVMS system service calls and socket library calls. The MBPPEX structure has a secondary error status word that should also be consulted. If a function call does not include an I/O Status block for secondary status information, the MBPPEX status variable should be evaluated. All of the API calls return a status, which should be tested first. If the function value status is successful (e.g., normal OpenVMS, odd values are successful), then the status structure should be tested when the I/O is complete depending if call is synchronous or asynchronous (calls with the “\_W” are synchronous).

The “cond” variable of the status block (mbp\_status.cond) can contain one of the following:

- Successful status: MBP\$SUCCESS (note this is a zero value)
- Socket error status
- Modbus Exception See Modbus Exception Codes, see section: 7.2
- API Library error See MBP Error Codes, see section: 7.3

Each transaction between the Modbus/TCP API and a slave device is at least a two step operation. The API issues a command to a slave device as the first step. The slave device must issue a response to that command. The response may be either the expected response, data or completion status in successful operations, or an exception response. An example of a case resulting in an exception response, would be sending a request to read holding registers which are not configured on the target device. The API would not fault the register address if it were valid, as defined in the Modicon documentation, but the slave device would issue an exception response since the register or registers did not exist on the slave. These exception response values are returned to the caller in the MBP\_STATUS structure used in most API calls. The exception codes defined in the “Open Modbus/TCP Specification are listed below.

#### **7.2 Modbus Exception Codes**

Exception Code (Hex)	Error Condition
01	<b>Illegal Function</b> , for the addressed slave. Refer to the specific device manual for further details.
02	<b>Illegal Data Address</b> , within the information field of the addressed slave. More specifically, the combination of starting register and count may be invalid at the slave device.

## MBP Error Codes

### Modbus Exception Codes

---

- 03        **Illegal Data Value**, in the information field for the addressed slave. This error is typically generated by a complex request such as read or write extended which requires additional data items to determine the target register file. The error is NOT based upon data values being stored into a register.
- 04        **Illegal Response Length**, indicates that the resulting response frame would exceed the maximum length allowed by the Modbus protocol. Typically this is the result of functions which generate a multi-part response such as Read General Reference and Write General Reference. *In the case of the Modbus/TCP API, you should not see this exception since the API only allows access to a single register file in each command.*
- 05        **Acknowledge**, is a response only generated in conjunction with programming commands issued to the slave device. Since the Modbus/TCP API doesn't support programming operations you should not see this exception code.
- 06        **Slave Device Busy**, is a response only generated in conjunction with programming commands issued to the slave device. Since the Modbus/TCP API doesn't support programming operations you should not see this exception code.
- 07        **Negative Acknowledge**, is a response only generated in conjunction with programming commands issued to the slave device. Since the Modbus/TCP API doesn't support programming operations you should not see this exception code.
- 08        **Memory Parity Error**, is a specialized response associated with the Read & Write General Reference operations which indicates that the extended memory file area failed to pass a consistency check.
- 0A        **Gateway Path Unavailable**, is a specialized response associates with the use of Modbus Plus Gateways. It is usually an indication that the gateway is misconfigured, and was unable to allocate a path to use for the request.
- 0B        **Gateway Target Device Failed To Respond**, is a specialized response associates with the use of Modbus Plus Gateways. It is usually an indication that the device is not present on the highway.

## **MBP Error Codes**

### **MBP Error Codes**

---

#### **7.3 MBP Error Codes**

The user should include the following statement in their link command if symbolic translation of the return status is desired: “mbtlib/include=(mbtshare\_msg)”.

**MBP\_ALRSUSPND** – Already suspended

A request was made to suspend the receipt of unsolicited messages by the calling process, but messages were already suspended.

**MBP\_BADPATH**-Path specified not allocated or out of range

A call to one of the functions contained a path number that was not within range. The path number must be between one and the number that was allocated by the MBP\_OPEN\_NET call.

**MBP\_BADSTARTADDR**-Invalid starting address

User has passed an invalid starting register or coil number, or has passed the value incorrectly. The starting register or coil must be a legal register number as stated in the “Modicon Ladder Logic Block Library User’s Guide”.

## **MBP Error Codes**

### **MBP Error Codes**

---

**MBP\_CRMTXFAIL**- Mutex creation failed

A primitive error indicating that the MBP\_OPEN\_NET function was unable to create a mutex object used internally for synchronization.

**MBP\_INVALIDARG**-Invalid argument

One of the user passed arguments is invalid, or has been passed incorrectly to the function.

**MBP\_INVALIDEFN**-Invalid event flag specified.

An event flag, or wait object, has been specified that is not valid. The MBP\_CRE\_SIGNAL\_OBJ function should be used to obtain a valid wait object.

**MBP\_INVALIDROUTE**- Invalid route specification.

All elements of the route array must be non-zero values in the range of 1 – 255<sub>10</sub>.

**MBP\_INVWAITTIME**- An invalid timeout interval was specified.

A valid timeout period must be greater than constant MBP\$K\_MIN\_WAIT. At this time the value is 100 milliseconds.

**MBP\_IOCANCEL**- I/O operation was cancelled.

A cancel request was issued for an outstanding request and the operation was successfully cancelled.

**MBP\_IPRCVFAIL**- A TCP/IP receive operation failed.

An operation failed due to a TCP/IP receive failure. This type of failure is most frequently caused by an underlying network error or remote device failure.

**MBP\_IPSENDFAIL**- A TCP/IP send operation failed.

An operation failed due to a TCP/IP send failure. This type of failure is most frequently caused by an underlying network error or remote device failure.

**MBP\_LNKCRFAIL**- TCP/IP connect failed.

As part of the initial operation to a target device, a TCP/IP connection must be made to the target. This status indicates that the connect request failed. This could be due to an incorrect route specification. It might also be due to the device being powered off or not having its TCP/IP parameters configured. If the target device is a node having a slave server, the slave server process may not be running at this time.

**MBP\_LOSTSYNC**- Message synchronization was lost.

An indication that a message received as part of an active operation contained a sequence number which was not the expected value. This is typically a lost packet on the network, or a slave device error.

## **MBP Error Codes**

### **MBP Error Codes**

---

**MBP\_MAPFAIL**-Unable to create process region

The MBP\_OPEN\_NET failed to create the group global section. The most probable error is insufficient privilege or quota. Examine the mbp\$l\_status in the MBPPEX structure.

**MBP\_MAPVPLCFAIL**-The application was unable to map the virtual PLC slave server holding register region.

The MBP\_MAP\_VPLC routine failed to map the specified virtual PLC. The most common error is related to the case sensitivity of the REGION\_NAME argument in the call. Examine the slave server region name that was specified and recognize that unless quotes are used to pass the slave server region name as a parameter, the region name will be lower case.

**MBP\_MBXCREFAIL**-Unable to create slave data mailbox

The register unsolicited was unable to create the mailbox. Examine the mbp\$l\_status in the MBPPEX structure for the reason. Most probable are insufficient privilege, or insufficient buffer quota.

**MBP\_MBXSIZ**E-Specified mailbox is too small

The specified mailbox size for slave reads (PLC master writes) is too small. It must be at least as large as the structure MASTER\_TO\_APPL\_BUF.

**MBP\_NAMEFAIL**- Unable to translate logical name for local mailbox.

During registration for unsolicited messages, the API creates a mailbox device for receiving the messages. It must translate a process logical name to obtain the device specification for the mailbox. For some reason that logical name cannot be translated at this time.

**MBP\_NOLINKSAVL**-No links blocks are available.

Each Master Path has a limit of MBP\$K\_MAXLINKS, currently 32, number of TCP/IP links. When the first command is issued to a slave device, over a designated master path, a TCP/IP link is created to that device. The link remains active until the slave device disconnects or when an error occurs. Once you have initiated commands to the maximum number of slave devices on the same path, you have consumed all available resources for that path. You may communicate with additional slave devices by using a different Master Path.

**MBP\_NOREGUNSOL**- Not registered for unsolicited messages.

This error is generated if a call is made to MBP\_SUSPEND\_UN SOLICITED or MBP\_RESUME\_UN SOLICITED prior to the application registering for unsolicited messaging.

**MBP\_NOSLAVE**-Caller did not allocate any slave paths

A request was made for unsolicited messaging, but no slave paths were previously allocated by the MBP\_OPEN\_NET call

## **MBP Error Codes**

### **MBP Error Codes**

---

**MBP\_NOTINITIALIZED**-MBPPEX structure not initialized

The user has not called MBP\_OPEN\_NET, the MBPPEX structure is corrupt, or the MBPPEX was passed incorrectly.

**MBP\_NOTSUSPND**- Unsolicited messaging not suspended on this path.

A request was made to resume unsolicited messaging on a path which has not had messaging suspended.

**MBP\_OPNOTACTIVE**- Operation not active on path.

A cancel was issued on a path which had no active operation.

**MBP\_PATHINUSE**-Path is in use

User has attempted to initiate operation on a path which has not completed the previous operation. This can occur when using the asynchronous versions of the API calls if synchronization objects are not used properly.

## MBP Error Codes

### MBP Error Codes

---

**MBP\_PIPEBROKEN**- TCP/IP Pipe broke.

An error condition indicating that an error has occurred in the underlying TCP/IP layers. The connection to the target slave device is broken. This can be caused by network errors or issues at the slave device.

**MBP\_SELECTFAIL**- TCP/IP Select failed.

An error condition indicating that an error has occurred in the underlying TCP/IP layers. The connection to the target slave device is broken. This can be caused by network errors or issues at the slave device.

**MBP\_SELECTTMO**- TCP/IP Select timed out.

A read operation on the path has timed out at the TCP/IP level. Each operation between a master and slave has a finite timeout period specified by the caller. If a single operation extends beyond this period a timeout is declared.

**MBP\_SENDFAIL**- TCP/IP Send failed.

At the TCP/IP level, a send operation failed. This is typically the result of a network condition or the failure of the remote device.

**MBP\_SLVASSIGNED**- Slave path already assigned for unsolicited messages

The caller has tried to register for unsolicited reads (master writes from a PLC) a second time on the same Slave Path. Only one registration may be executed on a single Slave Path.

**MBP\_THREADSIGFAIL**- Thread signal failed.

An internal failure of the API indicating that there was a failure to signal one of the I/O threads. This is not a recoverable error. It is likely that further operations on this path will fail. It might be possible for the application to use a different path.

**MBP\_THREADCREFAIL**- Thread creation failed.

An internal failure of the API indicating that there was a failure to create one of the I/O threads. This is not a recoverable error. The MBP\_OPEN\_NET call is responsible for the creation of all API related I/O threads.

**MBP\_TRANSTMO**- Unsolicited messaging transaction timed out

All unsolicited messaging transactions are handled by the local Slave Server process. Each transaction, such as registration, suspend, and resume, result in an acknowledge response to the calling application. This error indicates that the response was not received in an appropriate period of time.

**MBP\_UNSUPPORTED**- A request was made for an operation which is unsupported.

This error can occur in two cases. The first case is within the API library if the expanded region has somehow been corrupted. The second case is if the local Slave

## **MBP Error Codes**

### **MBP Error Codes**

---

Server receives a command, from a Master, which it is not able to support. The Slave Server supports only two operations at this time, Read Multiple Registers and Write Multiple Registers.

**MBP\_WRNGNMBRPATHS**-Wrong number of paths

The caller specified an incorrect number of paths. The number of paths was either less than 0, equal to 0, or greater than constant MBP\$K\_MAXPATHS, which is currently 16.

## **8 Appendix A Sample PLC MSTR**



## SamplePLC MSTR

### MSTR Example

---

NETWORK 0006 Segment: 01 These are the triggers for the MSTR instruction execution. The instruction can be triggered by a valid NODE and PATH initialization (see comment for network 5), manually with a contact from a forced coil, or repetitively with a timer providing delay between triggers. Additionally, the repeat function may be deactivated if a preset number of successive failures occur.

```

I NODE/PATH
I VALID
I TRIGGER
I
I
1+--] P[---+-----+----- ( )-
I 00999 ! ! 01001
I ! !
I ! !
I MANUAL ! !
I TRIGGER ! !
I ! !
2+-F] [---+
I 01000 ! !
I ! !
I MSTR STOP ! !
I MAX FAIL MSTR ! !
I EXCEEDED ACTIVE ! !
I ! !
3+--] \ [-----] \ [---+*-----*+
I 01012 01004 ! | 00015 |
I ! !
I ! MSTR |
I ! REPEAT |
I ! DLY ACCUM |
I ! |
4+ ++T1.0 +-
I | 40531 |
I *-----*

```

# SamplePLC MSTR

## MSTR Example

---

NETWORK 0007 Segment: 01

Whenever the MSTR instruction is to be executed (trigger occurs), the success or failure result of the last execution must be cleared (reset). After the result of the last MSTR instruction execution is cleared, the MSTR instruction execution is enabled by (DO MSTR) until it is completed (DONE). When an MSTR instruction execution terminates unsuccessfully, a counter is incremented, when termination is successful, the counter is reset. If the counter accumulator value reaches the preset value (999), the MSTR instruction is disabled via STOP MAX FAIL EXCEEDED.

```

I
I MSTR          MSTR          MSTR
I START        DONE GOOD      RESET
I
1+--] P[-----+--] [-----+----- ( ) -
I 01001        ! 01008        !          01002
I              !              !
I              ! MSTR         !
I              ! DONE BAD    !
I              !              !
2+          +--] [-----+
I              01009
I
I
I MSTR          MSTR          MSTR          MSTR
I DONE GOOD    DONE BAD      DONE          DO MSTR
I
3+--] \ [-----] \ [-----+--] \ [----- ( ) -
I 01008        01009        ! 01007        01003
I              !
I              !
I MSTR         !
I DO MSTR      !
I              !
4+--] [-----+
I 01003
I
I              MSTR STOP
I MSTR         MAX FAIL
I DONE BAD     EXCEEDED
I
5+--] P[-----*-----*----- ( ) -
I 01009        |00999      |          01012
I              |         |
I MSTR         |MSTR      |
I DONE GOOD    |FAILURE   |
I              |CNT ACCUM|
I              |         |
6+--] \ [-----+UCTR  +--
I 01008        |40511    |
I              *-----*

```

## SamplePLC MSTR

### MSTR Example

---

NETWORK 0008 Segment: 01

When an MSTR instruction execution terminates successfully, DONE GOOD is set (and sealed-in). When an MSTR instruction execution terminated unsuccessfully, DONE BAD is set (and sealed-in). When the MSTR instruction terminates, DONE is set.

```

I
I MSTR          MSTR          MSTR
I SUCCESS      RESET          DONE GOOD
I
1+--] P[-----+--]\[----- ( )-
I 01006       ! 01002          01008
I             !
I             !
I MSTR         !
I DONE GOOD   !
I             !
2+--] [-----+
I 01008
I
I
I MSTR          MSTR          MSTR
I FAILURE      RESET          DONEBAD
I
3+--] P[-----+--]\[----- ( )-
I 01005       ! 01002          01009
I             !
I             !
I MSTR         !
I DONE BAD    !
I             !
4+--] [-----+
I 01009
I
I
I             MSTR          MSTR
I             DONE GOOD    DONE
I
5+-----+--] [-----+----- ( )-
I             ! 01008     !          01007
I             !           !
I             !           !
I             ! MSTR     !
I             ! DONE BAD !
I             !           !
6+       +--] [-----+
I             01009
I

```



## **9 Appendix B Example VMSINSTALL**

## VMSINSTALL

### Example Product Install

---

#### 9.1 Example Product Install

The following is a typical install. The current version of OpenVMS and the Modbus Plus over TCP/IP for OpenVMS may not reflect the version shown here.

```
$ @sys$update:vmsinstal
  OpenVMS AXP Software Product Installation Procedure V7.1
```

It is 13-JAN-2000 at 16:50.

Enter a question mark (?) at any time for help.

```
%VMSINSTAL-W-NOTSYSTEM, You are not logged in to the SYSTEM account.
%VMSINSTAL-W-ACTIVE, The following processes are still active:
```

```
  ORA_INTRK_PMON
  ORA_INTRK_DBWR
  ORA_INTRK_ARCH
  ORA_INTRK_LGWR
  ORA_INTRK_SMON
  ORA_INTRK_RECO
  ORA_TNS5FA4195A
  DECW$TE_02FF
  Kevin
```

- \* Do you want to continue anyway [NO]? yes
- \* Are you satisfied with the backup of your system disk [YES]?
- \* Where will the distribution volumes be mounted: \$50\$mka500:

Enter the products to be processed from the first distribution volume set.

```
* Products: mbt010
* Enter installation options you wish to use (none):
Please mount the first volume of the set on $50$MKA500:.
* Are you ready? yes
%MOUNT-I-MOUNTED, MBT mounted on _$50$MKA500: (IPCALP)
The following products will be processed:
  MBT V1.0
```

Beginning installation of MBT V1.0 at 16:50

```
%VMSINSTAL-I-RESTORE, Restoring product save set A ...
%MOUNT-I-REMOVED, Product's release notes have been moved to SYS$HELP.
Moving MBTCP release notes to the SYS$HELP directory.
MODBUS/TCP Application Interface Library
```

11-Jan-2000

Copyright by:  
 IFACT Inc.  
 260 S. Campbell  
 Valparaiso, IN 46383  
 Ph: 219-464-7212  
All rights reserved

## VMSINSTALL

### Example Product Install

License Validation  
\* Enter node TCP/IP address for IPCAL3: : 216.176.131.154  
\* Enter node license number for 216.176.131.154:: ABCDEFGH

IpTouple: 216.176.131.154  
CK\_LIC Entered lic: ABCDEFGH  
License valid  
The distribution files have been restored from the  
saveset. The installation procedure is continuing...  
Your system or common device/directory: \$50\$DKB0:[SYS0.SYSCOMMON.]  
This product by default creates the directory:

[SYS0.SYSCOMMON.MBT010]

\* Use default directory location [Y]?  
Target Product location: \$50\$DKB0:[SYS0.SYSCOMMON.MBT010]  
\* Are these correct [Y]?

%MBT-I-COFFEEBREAK, answer section complete

Compiling and linking MBTCP message file.  
Linking the MBT\_SLAVE\_SERVER image.  
%MBT-I-COMMAND, Building startup command file MBT\_STARTUP.COM  
%MBT-I-DIRECTORY, Creating directory \$50\$DKB0:[SYS0.SYSCOMMON.MBT010]  
%VMSINSTAL-I-SYSDIR, This product creates system disk directory  
\$50\$DKB0:[SYS0.SYSCOMMON.MBT010].  
%CREATE-I-EXISTS, \$50\$DKB0:[SYS0.SYSCOMMON.MBT010] already exists  
%VMSINSTAL-I-SYSDIR, This product creates system disk directory  
\$50\$DKB0:[SYS0.SYSCOMMON.MBT010.EXAMPLES].  
%VMSINSTAL-I-SYSDIR, This product creates system disk directory  
\$50\$DKB0:[SYS0.SYSCOMMON.MBT010.LIB].  
%VMSINSTAL-I-SYSDIR, This product creates system disk directory  
\$50\$DKB0:[SYS0.SYSCOMMON.MBT010.DOC].  
%VMSINSTAL-I-SYSDIR, This product creates system disk directory  
\$50\$DKB0:[SYS0.SYSCOMMON.MBT010.PROD].  
%MBT-I-MOVING Marking files to move  
%MBT-I-COMPLETE, KITINSTAL.COM complete

This kit currently is not supplied with an IVP.  
The command file, MBT\_STARTUP.COM has been placed  
in directory SYS\$COMMON:[SYSMGR]. The following  
command line should be inserted in your system  
startup procedure SYSTARTUP\_VMS.COM located in  
directory SYS\$STARTUP.

\$ @SYS\$COMMON:[SYSMGR]MBT\_STARTUP

%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...  
Installation of MBT V1.0 completed at 16:51

Adding history entry in VMI\$ROOT:[SYSUPD]VMSINSTAL.HISTORY

Creating installation data file: VMI\$ROOT:[SYSUPD]MBT010.VMI\_DATA

**VMSINSTALL**  
**Example Product Install**

Enter the products to be processed from the next distribution volume set.

\* Products: exit

VMSINSTAL procedure done at 16:51

Directory MBT\_: [EXAMPLES]

TEST\_EXTENDED.C;1  
TEST\_MASK.C;1  
TEST\_MSTR.C;1  
TEST\_OPEN\_CLOSE.C;1  
TEST\_PROGRAM\_BLD\_C.COM;1  
TEST\_PROGRAM\_BLD\_F.COM;1  
TEST\_READS.C;1  
TEST\_READS.FOR;1  
TEST\_READ\_WRITE.C;1  
TEST\_UNSOLICITED.C;1  
TEST\_WRITER\_UNSOLICITED.C;1  
TEST\_WRITES.C;1

Total of 13 files.



**10 Appendix C iFIX access to Virtual PLC**

## iFIX Access to Virtual PLC

### Setting up the datablocks in iFIX

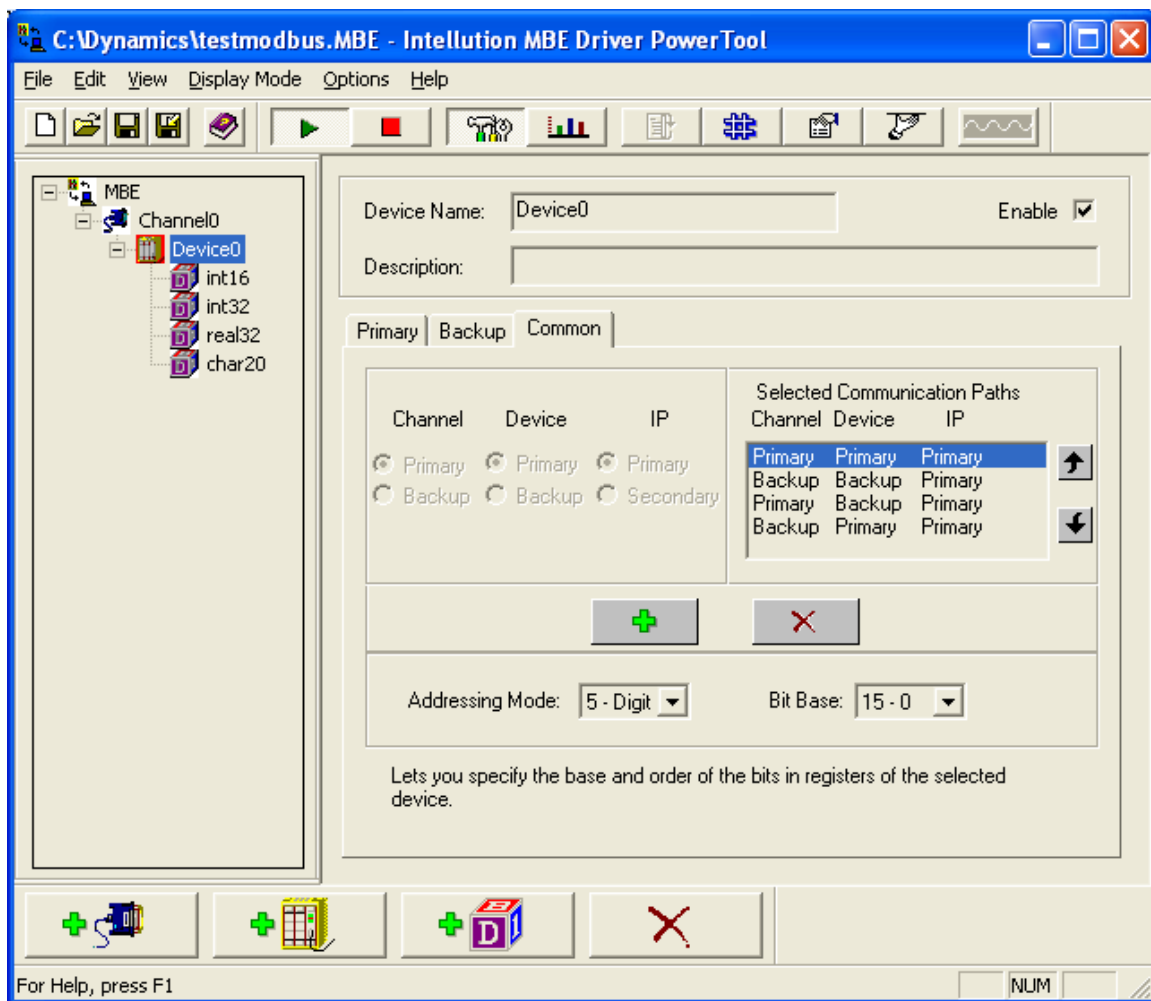
#### 10.1 Setting up the datablocks in iFIX

By using the IPACT Modbus over TCP/IP library and the Slave Server (Virtual PLC) one can display data from an OpenVMS host as if it were coming from a PLC into many popular HMI products. The applications on the OpenVMS host can deposit data into the holding registers which can be used to trigger data change scripts within the HMI's, alarming, or simply displayed on an HMI screen. In addition, the HMI can also update data within the holding registers on the OpenVMS host. The following shows how iFIX is configured to read the OpenVMS Virtual PLC in the same fashion as if it were reading a Modicon Quantum or other PLC.

The MBE Driver Power Tool is used to setup the datablocks in iFIX.

Common setting to all datablocks:

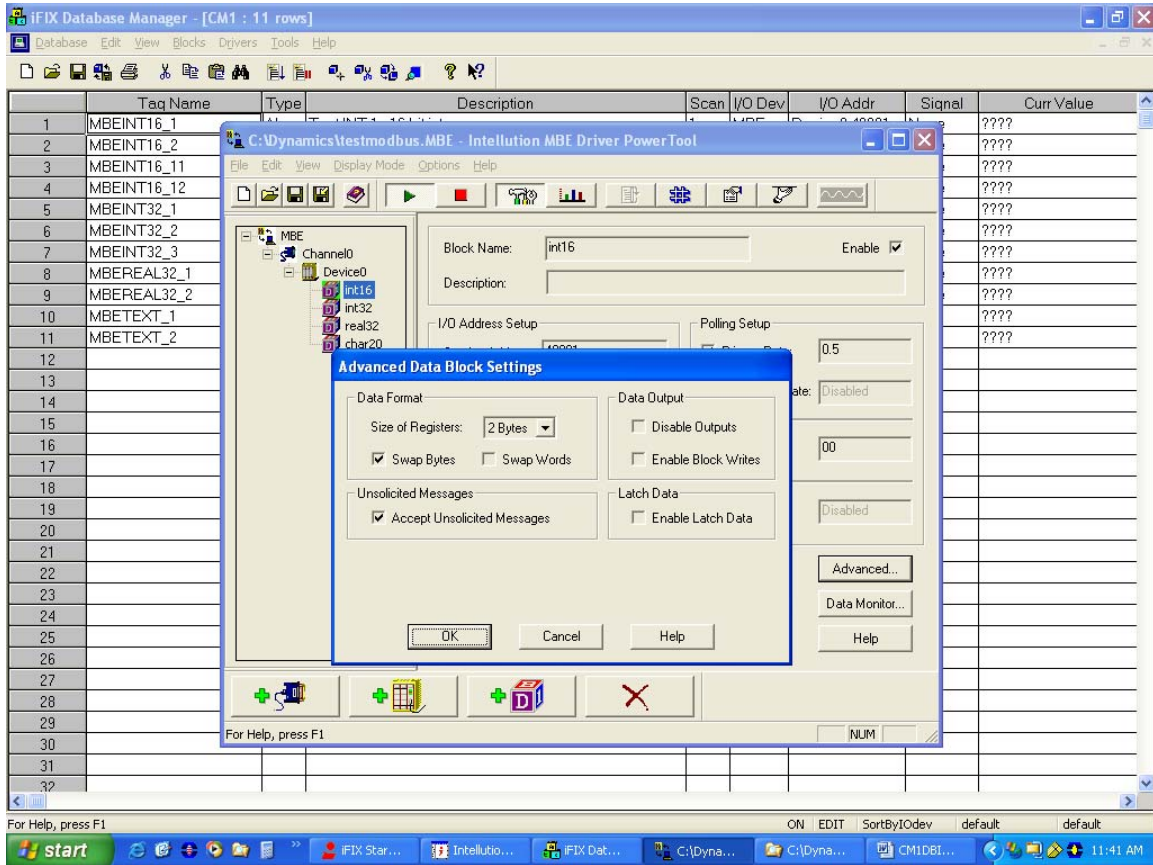
1. Select Device0 in the tree browser on the left and click the Common tab.
2. Click on the Addressing Mode to set it to 4, 5 or 6 digit.
3. Click on the Bit Base to set it to 0-15, 1-16, 15-0 or 16-1.



## iFIX Access to Virtual PLC Setting up the datablocks in iFIX

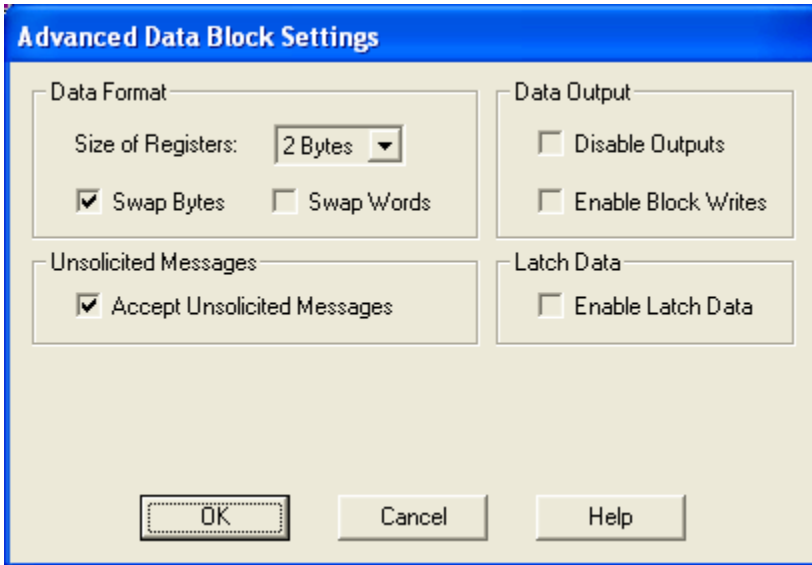
Individual datablock setting:

4. To set the data format click on one of the datablocks such as int16. Then on the bottom right select the Advance tab.
5. In the Data Format box there are fields to select the Size of the Registers which are 2 or 4, Swap Bytes and Swap Words.



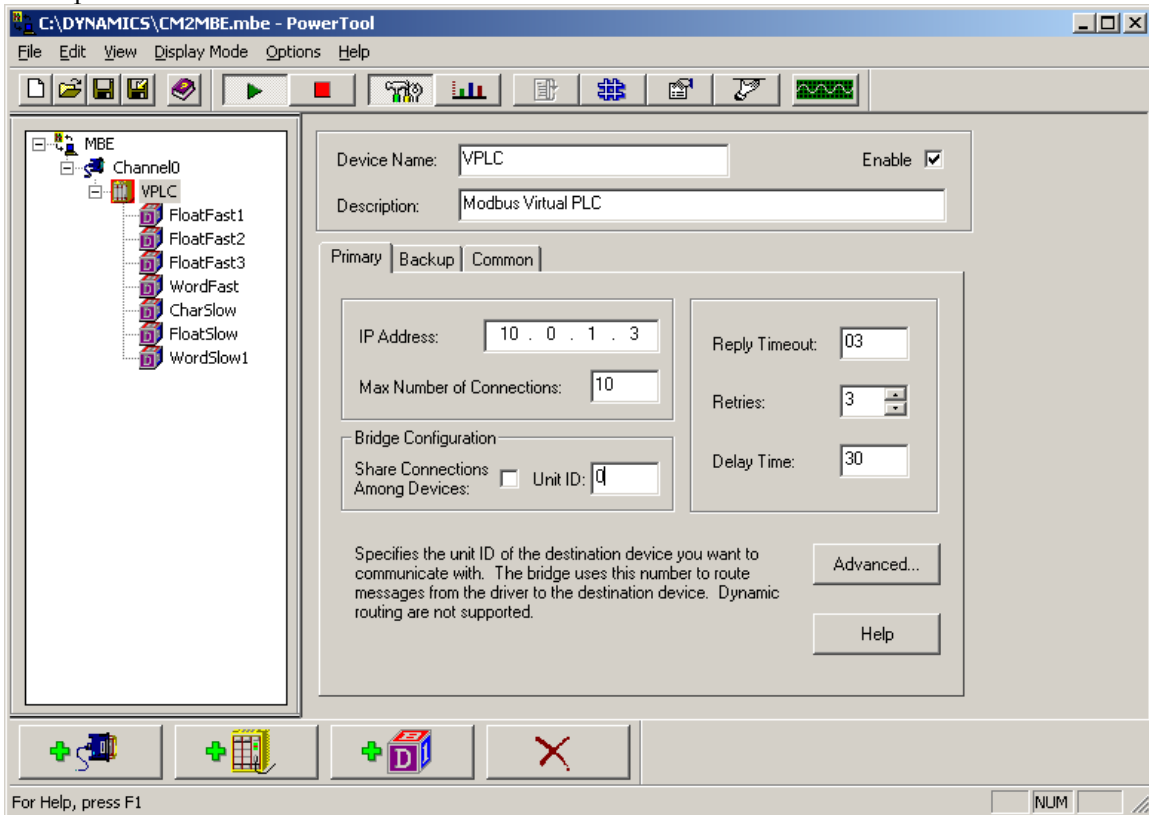
Configuring the Byte swapping option

## iFIX Access to Virtual PLC Selecting max number of sockets to use



### 10.2 Selecting max number of sockets to use

The iFIX configuration parameter “Max Number of Connections” determines how many socket connections may be created to the modbus server at any one time. If there are more blocks than connections, then some connected sockets will be used to send multiple blocks.



---

## **11 Appendix D iFIX Modbus tag Export**

## iFIX Modbus Tag Export

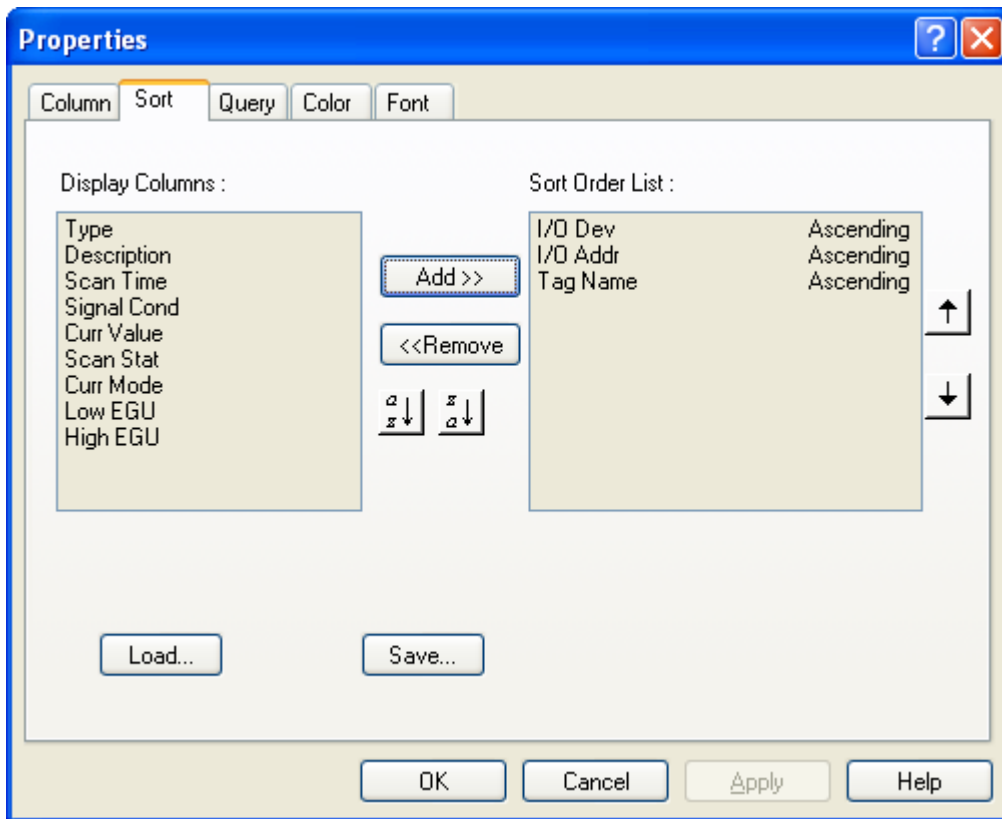
### Exporting Modbus tags from iFIX to a CSV file

---

#### 11.1 Exporting Modbus tags from iFIX to a CSV file

Exporting the Modbus tags from your iFIX application to a CSV file enables you to generate a CSV file containing the modbus tags. The CSV file may be used as a basis for generating an include file for use on the host node for mapping the virtual PLC registers. The iFIX export could be used as the Master Configuration for the common layout between the OpenVMS host virtual PLC common and the iFIX scan blocks.

1. In the Database Manager under the View, Properties menu selection a sort can be created to display only the tags that are configured to use the modbus+ MBE driver.
2. Click on the Sort Tab and add the I/O Dev column to the Sort Order List. Then move it to the top of the list. The Sort list can be saved or it can just be applied using the Apply button.

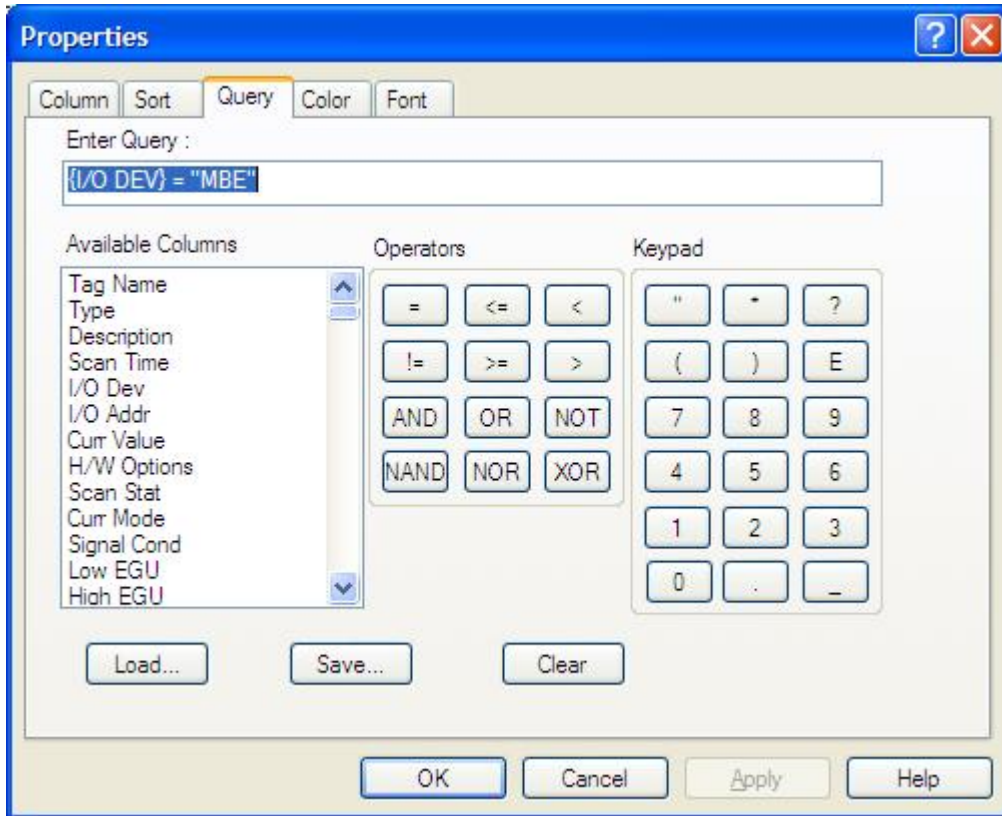


3. Then click on the Query tab and add/change the query to MBE so only the tags for the I/O Dev column will be shown. The Query can be saved or it can just be applied using the Apply button.

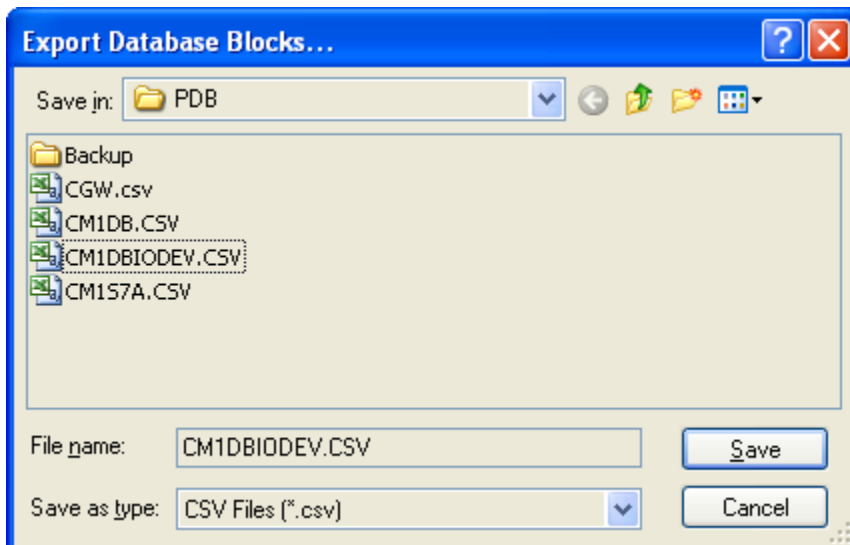
## iFIX Modbus Tag Export

### Exporting Modbus tags from iFIX to a CSV file

---



4. To export this database with only the modbus+ tags select from the menu bar Database, Export. The following popup appears.



5. Select or type in a File name and then Save.

# iFIX Modbus Tag Export

## Exporting Mobus tags from iFIX to a CSV file

The CSV file that is generated will look similar to this:

```
[NodeName : CM1,Database : CM1DB,File Name : C:\DYNAMICS\LOGANSQL\PDB\CM1DBIODEV.CSV,Date : 12/15/2004,Time : 9:33:34 AM]

[BLOCK TYPE,TAG,NEXT BLK,DESCRIPTION,INITIAL SCAN,SCAN TIME,SMOOTHING,I/O DEVICE,H/W OPTIONS,I/O ADDRESS,SIGNAL CONDITIONING,LOW EGU LIMIT,HIGH EGU LIMIT,EGU TAG,INITIAL A/M STATUS,ALARM ENABLE,ALARM AREA(S),LO LO ALARM LIMIT,LO ALARM LIMIT,HI ALARM LIMIT,HI HI ALARM LIMIT,ROC ALARM LIMIT,DEAD BAND,ALARM PRIORITY,ENABLE OUTPUT,SECURITY AREA 1,SECURITY AREA 2,SECURITY AREA 3,ALARM AREA 1,ALARM AREA 2,ALARM AREA 3,ALARM AREA 4,ALARM AREA 5,ALARM AREA 6,ALARM AREA 7,ALARM AREA 8,ALARM AREA 9,ALARM AREA 10,ALARM AREA 11,ALARM AREA 12,ALARM AREA 13,ALARM AREA 14,ALARM AREA 15,USER FIELD 1,USER FIELD 2,ESIG TYPE,ESIG ALLOW CONT USE,ESIG XMPT ALARM ACK,ESIG UNSIGNED WRITES]

!A_NAME,A_TAG,A_NEXT,A_DESC,A_ISCAN,A_SCANT,A_SMOOTH,A_IODV,A_IOHT,A_LOAD,A_IOSC,A_ELO,A_EHI,A_EGUDESC,A_IAM,A_IENAB,A_ADI,A_LOLO,A_LO,A_HI,A_HIHI,A_ROC,A_DBAND,A_PRI,A_EOUT,A_SA1,A_SA2,A_SA3,A_AEA1,A_AEA2,A_AEA3,A_AEA4,A_AEA5,A_AEA6,A_AEA7,A_AEA8,A_AEA9,A_AEA10,A_AEA11,A_AEA12,A_AEA13,A_AEA14,A_AEA15,A_ALMEXT1,A_ALMEXT2,A_ESIGTYPE,A_ESIGCONT,A_ESIGACK,A_ESIGTRAP!

AI,MBEINT16_1,,Test INT 1 - 16 bit
integer,ON,1,0,MBE,,Device0:40001,None,0,"100,000",,AUTO,ENABLE,NONE,0,0,"100,000","100,000",0,"5,000",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

AI,MBEINT16_2,,Test INT 2 - 16 bit
integer,ON,1,0,MBE,,Device0:40002,None,0,"100,000",,AUTO,ENABLE,NONE,0,0,"100,000","100,000",0,"5,000",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

AI,MBEINT16_11,,Test INT 3 - 16 bit
integer,ON,1,0,MBE,,Device0:40011,None,0,"100,000",,AUTO,ENABLE,NONE,0,0,"100,000","100,000",0,"5,000",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

AI,MBEINT16_12,,Test INT 4 - 16 bit
integer,ON,1,0,MBE,,Device0:40012,None,0,"100,000",,AUTO,ENABLE,NONE,0,0,"100,000","100,000",0,"5,000",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

AI,MBEINT32_1,,Test INT 1 - 32 bit
integer,ON,1,0,MBE,,Device0:40101,None,0,"100,000",,AUTO,ENABLE,NONE,0,0,"100,000","100,000",0,"5,000",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

AI,MBEINT32_2,,Test INT 2 - 32 bit
integer,ON,1,0,MBE,,Device0:40103,None,0,"100,000",,AUTO,ENABLE,NONE,0,0,"100,000","100,000",0,"5,000",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

AI,MBEINT32_3,,Test INT 3 - 32 bit
integer,ON,1,0,MBE,,Device0:40105,None,0,"100,000",,AUTO,ENABLE,NONE,0,0,"100,000","100,000",0,"5,000",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

AI,MBEREAL32_1,,Test Real 1 - 32
bit,ON,1,0,MBE,,Device0:40201,None,0,"100,000.00",,AUTO,ENABLE,NONE,0,0,"100,000.00","100,000.00",0,"5,000.00",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

AI,MBEREAL32_2,,Test Real 2 - 32
bit,ON,1,0,MBE,,Device0:40203,None,0,"100,000.00",,AUTO,ENABLE,NONE,0,0,"100,000.00","100,000.00",0,"5,000.00",L,YES,NONE,NONE,NONE,ALL,,,,,,,,,,,,,,,,,,,,,NONE,YES,NO,REJECT

[BLOCK TYPE,TAG,INITIAL SCAN,SCAN TIME,INITIAL A/M STATUS,I/O DEVICE,H/W OPTIONS,I/O ADDRESS,MESSAGE LENGTH,ALARM ENABLE,ALARM AREA(S),EVENT MSG,SECURITY AREA 1,SECURITY AREA 2,SECURITY AREA 3,ALARM AREA 1,ALARM AREA 2,ALARM AREA 3,ALARM AREA 4,ALARM AREA 5,ALARM AREA 6,ALARM AREA 7,ALARM AREA 8,ALARM AREA 9,ALARM AREA 10,ALARM AREA 11,ALARM AREA 12,ALARM AREA 13,ALARM AREA 14,ALARM AREA 15,USER FIELD 1,USER FIELD 2,DESCRIPTION,ESIG TYPE,ESIG ALLOW CONT USE,ESIG XMPT ALARM ACK,ESIG UNSIGNED WRITES],,,,,,,,,,,,,

!A_NAME,A_TAG,A_ISCAN,A_SCANT,A_IAM,A_IODV,A_IOHT,A_LOAD,A_LEN,A_IENAB,A_ADI,A_EVENT,A_SA1,A_SA2,A_SA3,A_AEA1,A_AEA2,A_AEA3,A_AEA4,A_AEA5,A_AEA6,A_AEA7,A_AEA8,A_AEA9,A_AEA10,A_AEA11,A_AEA12,A_AEA13,A_AEA14,A_AEA15,A_ALMEXT1,A_ALMEXT2,A_DESC,A_ESIGTYPE,A_ESIGCONT,A_ESIGACK,A_ESIGTRAP!

TX,MBETEXT_2,ON,1,AUTO,MBE,,Device0:40311,20,ENABLE,NONE,DISABLE,NONE,NONE,NONE,ALL,,,,,,,,,,,,,Test Text 2 - Length 20,NONE,YES,NO,REJECT
```

**iFIX Modbus Tag Export**  
**Exporting Modbus tags from iFIX to a CSV file**

---

```
TX,MBETEXT_1,ON,1,AUTO,MBE,,Device0:40301,20,ENABLE,NONE,DISABLE,NONE,NONE,NONE,ALL,,,,,,,,,
,,,,,,,,,Test Text 1 - Length 20,NONE,YES,NO,REJECT

[-----End of Block List-----]
-----]
```

## **12 Appendix E WonderWare Intouch & Virtual PLC**

## Wonderware InTouch & Virtual PLC Configuring the Access Name

### 12.1 Configuring the Access Name

In WindowMaker select the “Special” tab and then “Access Names...” from the drop down menu. Select “Add” to configure a new name. The Access name is what Intouch uses when configuring a tag. The Topic Name is the name created in the I/O server configuration. The Application Name is the name of the I/O Server.

The screenshot shows the "Modify Access Name" dialog box. It has a blue title bar with the text "Modify Access Name". Below the title bar, there are several fields and options:

- Access:** A text box containing "VPLC".
- Node Name:** An empty text box.
- Application Name:** A text box containing "MBENET".
- Topic Name:** A text box containing "VPLC".
- Which protocol to use:** A group box containing three radio buttons: "DDE" (selected), "SuiteLink", and "Message Exchange".
- When to advise server:** A group box containing two radio buttons: "Advise all items" and "Advise only active items" (selected).
- Buttons:** "OK" and "Cancel" buttons are located on the right side of the dialog.

### 12.2 Configuring the MBENET I/O Server

In the I/O Server select the “Configure” tab and then “Topic Definition...” from the drop down menu. Select “New...” button and then enter in the same Topic name that was configured above in Intouch. Enter in the IP address the Slave Device Type (PLC). For 5-digit addressing select the 584/984 PLC. Use the default settings for the rest of the configuration.

## Wonderware InTouch & Virtual PLC Configuring the MBENET I/O Server

---

**MBENET Topic Definition** ✕

Topic Name:

IP Address:

Device Index or Unit\_ID:

Slave Device Type:  ▼

Use Concept Data Structures

Communication Channels

Unsolicited Messages

String Variable Style

Full length  
 C style  
 Pascal style

Register Type

Binary  
 BCD

Block I/O Sizes

Coil Read:  Register Read:

Coil Write:  Register Write:

Update Interval:  msec Reply Timeout:  sec

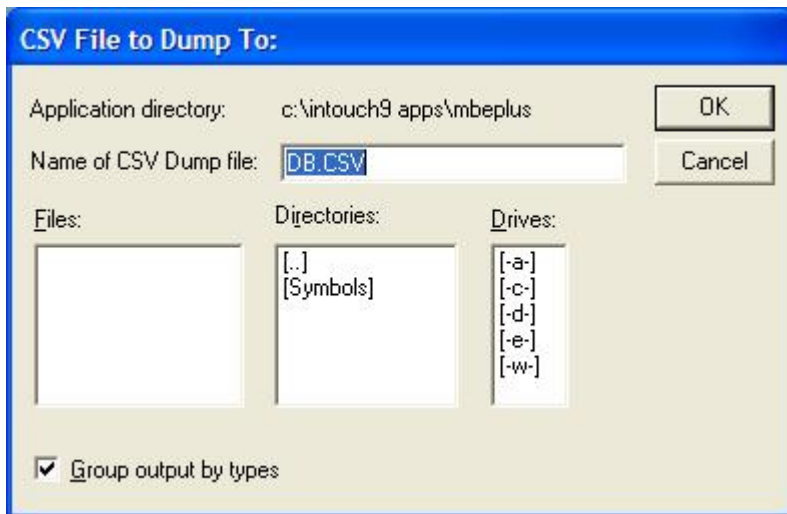
## **13 Appendix F InTouch Modbus Tag Export**

## InTouch Modbus Tag Export Configuring the Access Name

### 13.1 Configuring the Access Name

Exporting the Modbus tags from your Intouch application to a CSV file enables you to generate a CSV file containing the modbus tags. The CSV file may be used as a basis for generating an include file for use on the host node for mapping the virtual PLC registers. The Intouch export could be used as the Master Configuration for the common layout between the OpenVMS host virtual PLC common and Intouch.

Select DBDump from the Intouch – Application Manager after selecting the project that the CSV file will be made from. Then select the name and the directory as shown below. After the file is made use MS Excel to sort out the Modbus tags.



## **14 Appendix G TCP Communication Issues**

## **TCP/IP Communications Issues**

### **TCP Communication Configuration**

#### **14.1 TCP Communication Configuration**

The Slave Server application uses sockets for the connection to slave systems. The keepalive feature of TCP is used to cause the sockets to close should the link to the slaves disappear. The following parameters are used by OpenVMS to determine how long it will take before the sockets are disconnected. Note that the values vary for different versions of OpenVMS TCP/IP Services. This results in sockets staying around for about 675 seconds on some versions and 7800 seconds on RPC compliant versions.

##### **TCP\_KEEPIDLE**

When the SO\_KEEPAALIVE option is enabled, TCP sends a keepalive probe to the remote system of a connection that has been idle for a period of time. If the remote system does not respond to the keepalive probe, TCP retransmits a keepalive probe for a certain number of times before a connection is considered to be broken. TCP\_KEEPIDLE specifies the number of seconds before TCP will send the initial keepalive probe. The default value for TCP\_KEEPIDLE is an integer value between 1 and  $n$ , where  $n$  is the value for the systemwide parameter `tcp_keeppidle`. The default value for `tcp_keeppidle`, specified in half-second units, is 150 (75 seconds). To display the values of the systemwide parameters, enter the following command at the system prompt:

```
TCPIP> sysconfig -q inet
```

The default value for TCP\_KEEPIDLE is 75 seconds.

*NOTE: TCP\_KEEPIDLE has changed on different versions of OpenVMS TCP/IP Services. For example, on version 5.1 this value is 150 (75 seconds), while in later versions it was changed to 2 hours, which is the RPC specification.*

##### **TCP\_KEEPINIT**

If a TCP connection cannot be established within a period of time, TCP will time out the connection attempt. The default timeout value for this initial connection establishment is 75 seconds. The TCP\_KEEPINIT option specifies the number of seconds to wait before the connection attempt times out. For passive connections, the TCP\_KEEPINIT option value is inherited from the listening socket. The value of TCP\_KEEPINIT is an integer between 1 and  $n$ , where  $n$  is the value for the systemwide parameter `tcp_keeppinit`. The default value of the systemwide parameter `tcp_keeppinit`, specified in half-second units, is 150 (75 seconds). To display the values of the systemwide parameters, enter the following command at the system prompt:

```
TCPIP> sysconfig -q inet
```

The TCP\_KEEPINIT option does not require the SO\_KEEPAALIVE option to be enabled.

##### **TCP\_KEEPCNT**

When the SO\_KEEPAALIVE option is enabled, TCP sends a keepalive probe to the remote system of a connection that has been idle for a period of time. If the remote system does not respond to the keepalive probe, TCP retransmits a keepalive probe for a certain number of times before a connection is considered to be broken. The TCP\_KEEPCNT option specifies the maximum number of keepalive probes to be sent. The value of TCP\_KEEPCNT is an integer value between 1 and  $n$ , where  $n$  is the value of the systemwide `cp_keeppcnt` parameter. The default value for for the systemwide parameter, `tcp_keeppcnt`, is 8. To display the values of the systemwide parameters, enter the following command at the system prompt:

## **TCP/IP Communications Issues**

### **TCP Communication Configuration**

---

```
TCPIP> sysconfig -q inet
```

The default value for TCP\_KEEPCNT is 8.

#### **TCP\_KEEPINTVL**

When the SO\_KEEPALIVE option is enabled, TCP sends a keepalive probe to the remote system on a connection that has been idle for a period of time. If the remote system does not respond to a keepalive probe, TCP retransmits the keepalive probe after a period of time. The default value for this retransmit interval is 75 seconds. The TCP\_KEEPINTVL option specifies the number of seconds to wait before retransmitting a keepalive probe. The value of the TCP\_KEEPINTVL option is an integer between 1 and  $n$ , where  $n$  is the value of the systemwide parameter `tcp_keepintvl` which is specified in half-second units. The default value for the systemwide parameter `tcp_keepintvl` is 150 (75 seconds).

To display the values of the systemwide parameters,

enter the following command at the system prompt:

```
$ sysconfig -q inet
```

#### **TCP PROBE TIMER**

This parameter can be viewed using the following command within TCPIP:

```
TCPIP> show protocol/parameter tcp
```

This parameter determines how often a socket is checked to make sure it is alive. A value of 5 seconds is a reasonable value for a locally connected Ethernet. If your system has a default value, it can be changed with the following command:

```
TCPIP> set protocol tcp/probe_timer=5
```

## **15 Appendix H Modbus over TCP/IP** **Compliant Devices**

## **TCP/IP Compliant Devices Compliant Devices**

### **15.1 Compliant Devices**

This API library will allow OpenVMS users to reference other Modbus over TCP/IP devices that conform to the Modbus on TCP/IP standard (see [www.modbus.org](http://www.modbus.org)). The library was designed to communicate with Modicon PLCs but has been successfully used to communicate with GE-FANUC PAC processeors and some temperature devices.

#### **15.1.1 GE FANUC PAC processors**

The GE FANUC PAC processors can support Modbus TCP/IP from its Ethernet port. However, the PAC firmware must be at the correct level to support Modbus over TCP/IP.

#### **15.1.2 Rosemont FIM 3420**

The Rosemont's FIM 3420. (Fieldbus Interface Module, <http://www.emersonprocess.com/rosemount/document/man/00809-0100-4023.pdf>) successfully responds using holding register reads. Both the integer and floating point temperature values along with the clock are readable.

#### **15.1.3 Thermo-Fisher X-ray**

The Thermo-Fisher is planning support for their X-ray in 2010.

# Index

---

Asynchronous, 15  
Byte Order, 12  
Byte Swapping, 12, 64  
Data Representation, 12  
Debug Library, 5  
Dedicated Slave Paths, 12  
Exit Handler, 33  
GE FANUC PAC, 115  
Host, 13  
iFIX, 18  
InTouch, 18  
Linker, 5  
Linking Requirements, 5  
Master Paths, 9  
MASTER\_TO\_APPL\_BUF, 21, 45, 46, 83  
MBP\$K\_MAXLINKS, 83  
MBP\$K\_MIN\_WAIT, 82  
MBP\_BADPATH, 36, 47, 81  
MBP\_BADSTARTADDR, 81  
mbp\_bufdef.h, 46  
MBP\_CLOSE\_NET  
    Defined, 24  
MBP\_CRE\_SIGNAL\_OBJ  
    Defined, 25  
MBP\_CRMTXFAIL, 82  
MBP\_DEL\_SIGNAL\_OBJ  
    Defined, 26  
MBP\_FORCE\_SINGLE\_COIL  
    Defined, 27  
MBP\_INVALIDARG, 64, 82  
MBP\_INVALIDEFN, 82  
MBP\_INVALIDROUTE, 82  
MBP\_INVLPATH, 43, 61, 65  
MBP\_INVWAITTIME, 82  
MBP\_IOCANCEL, 82  
MBP\_IPRCVFAIL, 82  
MBP\_IPSENFFAIL, 82  
MBP\_LNKCRFAIL, 82  
MBP\_LOSTSYNC, 82  
MBP\_MAP\_VPLC, 20, 21  
    Defined, 30  
MBP\_MAPFAIL, 83  
MBP\_MBPEXINVL, 39, 43, 64  
MBP\_MBXCREFAIL, 83  
MBP\_MBXFAIL, 51  
MBP\_MBXSIZE, 83  
MBP\_NAMEFAIL, 83  
MBP\_NOLINKSAVL, 83  
MBP\_NOREGUNSOL, 53, 83  
MBP\_NOSLAVE, 51, 53, 55, 83  
MBP\_NOSUCHPATH, 53, 55  
MBP\_NOTINITIALIZED, 36, 47, 84  
MBP\_NOTSUSPND, 53, 84  
MBP\_OPEN\_NET  
    Defined, 32  
MBP\_OPNOTACTIVE, 84  
MBP\_PATHINUSE, 36, 39, 44, 47, 61, 65, 84  
MBP\_PIPEBROKEN, 85  
MBP\_READ\_EXTENDED  
    Defined, 37  
MBP\_READ\_EXTENDEDW  
    Defined, 37  
MBP\_READ\_REGISTERS  
    Defined, 41  
MBP\_READ\_REGISTERSW  
    Defined, 41  
MBP\_READ\_UNSOLICITED, 50  
MBP\_REGISTER\_UNSOLICITED, 20, 33, 46  
MBP\_RESUME\_UNSOLICITED  
    Defined, 52  
MBP\_SELECTFAIL, 85  
MBP\_SENDFAIL, 85  
MBP\_SLVASSIGNED, 85  
MBP\_SSELECTTMO, 85  
MBP\_STATUS, 13, 79  
MBP\_SUSPEND\_UNSOLICITED  
    Defined, 54  
MBP\_THREADCREFAIL, 85  
MBP\_THREADSIGFAIL, 85  
MBP\_TRANSTMO, 85  
MBP\_UNSUPPORTED, 85  
MBP\_VPLB\_ACCESS, 21  
MBP\_VPLC\_ACCESS  
    Defined, 56  
MBP\_WRITE\_EXTENDED  
    Defined, 58  
MBP\_WRITE\_EXTENDEDW  
    Defined, 58  
MBP\_WRNGNMBRPATHS, 86  
MBPDEF\_C, 13  
MBPDEF\_F, 13  
MBT\_Logical, 5  
Modbus/TCP, 7  
Object Library, 5  
Path, 8  
Path (defined), 2  
Path Allocation, 9  
PLC Register Addressing, 7  
Privilege, 51  
Process Expanded Region, 14  
Process Quotas, 6  
Process Region, 33  
Quotas, 51  
ReadReg utility, 70  
Required Privileges, 6  
Route, 12

# Index

---

Slave Path, 45  
Slave Paths, 10, 12  
Slave Server, 18  
Status Block, 13  
Synchronous, 15  
Thread, 15  
THREADCP, 6  
Timeout, 14  
Unit, 12  
Virtual PLC, 18  
WriteReg utility, 74