

MBLogic

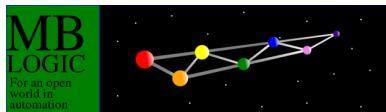
for an open world in automation

User Manual

By

MBLogic & Python Fan

February 23, 2013



About

Overview:

This software system is known as MBLogic, and has the following capabilities:

- Has a built in soft logic (PLC) system.
 - Has a built in web based HMI.
 - Acts as a communications server (master).
 - Acts as a communications client (slave).
 - Can be both server and multi-client system simultaneously.
 - Supports multiple protocols simultaneously.
 - Communications can be extended with user-written modules.
 - Has a built in web based status and control system.
 - Has extensive on line documentation.
-

System Requirements:

The software has the following system requirements:

- Will run on any computer capable of supporting Python 2.6 and the Twisted networking framework.
 - Hard drive space requirements for the basic software (excluding Python, Twisted and user application files) is approximately ten megabytes.
 - The required speed for the computer will depend upon the size of the soft logic program, the number of communications connections and type of protocols used, and the scan and polling speeds used. An average PC should be able to support over 100 connections. Several dozen connections should be able to be supported while maintaining a low CPU load.
-

Licensing:

This software licensed under the GNU General Public License (GPL) and is free software. See the included license document for more details as to the applicable version and terms of the license. You may copy and distribute this software freely provided you adhere to the terms of the license.

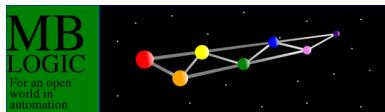
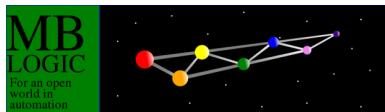
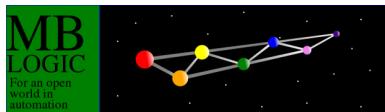


Table of Contents

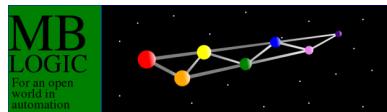
1	Introduction	1
1.1	Documentation	1
1.1.1	Overview	1
2	General.....	2
2.1.1	Overview	2
2.1.2	On Line Help.....	3
2.1.3	Help - Data Table.....	5
2.2	Creating an Application.....	7
2.2.1	Overview	7
2.2.2	Create the Application	7
2.3	File Locations.....	10
2.3.1	Overview	11
2.3.2	System Files Location	11
2.3.3	Configuration Files and Soft Logic Program.....	11
2.3.4	HMI Files.....	12
2.3.5	User Help Pages	13
2.3.6	Generic Clients	13
2.4	System Capacity	13
2.4.1	Overview	13
2.4.2	Maximum System Capacity.....	14
2.5	User Created Help Pages.....	15
2.5.1	Overview	15
2.5.2	Web Pages.....	15
2.5.3	Web Site Configuration.....	16
2.5.4	Web Site File Location.....	16
3	Status Status	17
3.1	Overview	17
3.2	System Status.....	17
3.2.1	Overview	17



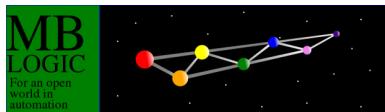
3.2.2	General System Parameters.....	18
3.2.3	Subsystem Status	18
3.2.4	Static Web Status Page	19
3.3	Configure Summary	20
3.3.1	Overview	20
3.3.2	Configurations.....	20
3.3.3	HMI Web Pages.....	21
3.4	Edit Configurations.....	22
3.4.1	Overview	22
3.4.2	Enabling Editing.....	22
3.4.3	Input Forms.....	23
3.4.4	Validating Input.....	24
3.4.5	Saving Data.....	24
3.4.6	Server Validation.....	25
3.4.7	When Changes Take Effect	26
3.5	Configure Server Communications	27
3.5.1	Overview	27
3.5.2	Server ID	27
3.5.3	Servers.....	28
3.5.4	Data Table Expanded Register Map	29
3.6	Configure Client Communications	30
3.6.1	Overview	30
3.6.2	TCP Clients.....	30
3.6.3	Generic Clients	36
3.7	Configure HMI.....	41
3.7.1	Overview	41
3.7.2	HMI Parameters	42
3.7.3	Editing Parameters.....	44
3.8	Configure Soft Logic IO.....	47
3.8.1	Overview	47
3.8.2	Soft Logic Parameters	48



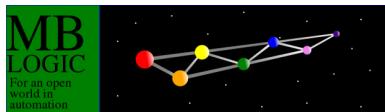
3.8.3	Edit Parameters.....	49
3.9	Soft Logic Program	51
3.9.1	Overview	51
3.9.2	Editing a Subroutine in IL	52
3.9.3	Adding and Deleting a Subroutine	54
3.10	Control	55
3.10.1	Overview	55
3.10.2	Configuration Reload	55
3.10.3	System Shutdown	56
3.10.4	System Restart	56
3.11	Monitor Data and Program.....	58
3.11.1	Overview	58
3.11.2	Monitor Data Table	58
3.11.3	Monitor Ladder	60
3.12	System Monitor.....	63
3.12.1	Overview	63
3.12.2	Program Run Status	63
3.12.3	Communications	64
3.12.4	Communications Monitor	65
3.13	Cross Reference	67
3.13.1	Overview	67
3.13.2	Display Order	68
4	Communications	69
4.1	Overview:	69
4.1.1	Communications Help Topics:.....	69
4.2	Server Communications Configuration.....	71
4.2.1	Overview	71
4.2.2	Configuration Overview	71
4.2.3	System Configuration.....	72
4.2.4	Server Configuration	74
4.2.5	Complete Example	76



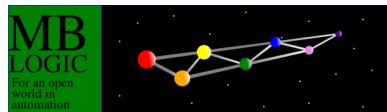
4.2.6	Loop Back Sample	77
4.3	Client Communications Configuration	78
4.3.1	Overview	78
4.3.2	Configuration Overview	78
4.3.3	TCP Client Connections	79
4.3.4	Generic Clients	83
4.3.5	Fault Configuration	88
4.3.6	Complete Example	89
4.3.7	Loop Back Sample	91
4.4	Generic Client Protocols	92
4.4.1	Overview	92
4.4.2	Supported Protocols	92
4.4.3	Generic Client Configuration.....	92
4.5	Communications Errors.....	93
4.5.1	Overview	93
4.5.2	Types of Errors	93
4.5.3	Status and Error Codes.....	93
4.5.4	Error Reporting	95
4.5.5	Resetting Error Codes	96
4.5.6	Configuration	97
4.6	Communications Monitoring and Trouble Shooting	98
4.6.1	Overview	98
4.7	Expanded Register Map	99
4.7.1	Overview	99
4.7.2	Expanded Map Sizes.....	99
4.7.3	Disabling the Expanded Register Map	99
4.7.4	Supported Protocols	100
4.7.5	Offset Calculations	100
4.7.6	Error Handling	101
4.7.7	Example Using Modbus/TCP Server Protocol	101
4.8	Modbus Support	103



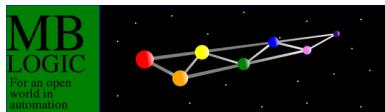
4.8.1	Overview	103
4.9	MB-REST.....	104
4.9.1	Overview	104
4.9.2	Web Service Concepts.....	104
4.9.3	Reading Data	105
4.9.4	Writing Data	106
4.9.5	Server Response.....	106
4.9.6	XML Document Tags	108
4.9.7	Data Formats.....	108
4.9.8	HTTP Errors	109
4.9.9	Modbus Exceptions.....	109
4.9.10	Limits on Message Size	109
4.9.11	Protocol Speed and Overhead	110
4.10	HMI Protocol	111
4.10.1	Overview	111
4.10.2	HMI Servers and Services.....	111
4.10.3	Address Tags	112
4.10.4	Alarms	113
4.10.5	Events.....	114
4.10.6	Alarm and Event Zones	115
4.10.7	Alarm and Event Serial Numbers	116
4.10.8	Supporting Multiple HMI Systems	116
4.10.9	Logging Alarm History and Events	116
4.11	ERP Protocol.....	117
4.11.1	Overview	117
4.11.2	Web Service Protocols	117
4.11.3	Basic Message Format	117
4.11.4	Read Command.....	118
4.11.5	Write Command.....	119
4.11.6	Combined Read/Write	119
4.11.7	Read and Write Errors.....	120



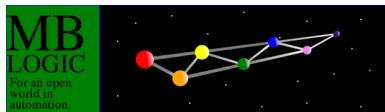
4.11.8	Configuration	121
4.11.9	Creating an ERP Client Interface	121
4.11.10	Example Client	122
4.12	Generic Clients	125
4.12.1	Overview	125
4.12.2	Generic Client Server Interface	125
4.12.3	Generic Client Start and Stop Sequence	126
4.12.4	File Locations.....	127
4.12.5	Protocols and Message Transports.....	127
4.13	Generic Client Framework	128
4.13.1	Overview	128
4.13.2	Framework API.....	128
4.13.3	User Protocol Class	129
4.13.4	Signal Handlers.....	130
4.13.5	Example.....	130
4.14	Generic Client User Protocol Class.....	134
4.14.1	Overview	134
4.14.2	Import Modules	134
4.14.3	Client Protocol Class.....	134
4.15	RSS Feed.....	140
4.15.1	Overview	140
4.15.2	Addressing.....	140
4.15.3	Configuration	140
4.15.4	The RSS Template.....	142
4.15.5	Event Texts File	144
4.15.6	Loading and Changing the RSS Messages	145
4.15.7	Server Configuration	145
4.16	Modbus Basics	146
4.16.1	Overview	146
4.16.2	Types of Modbus.....	146
4.16.3	Communications	146



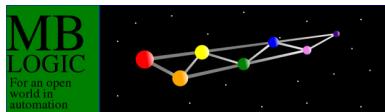
4.16.4	Modbus Data Representation:.....	149
4.16.5	Modbus Commands, or "Functions"	151
4.16.6	Modbus Errors	152
5	Soft Logic.....	154
5.1	Overview	154
5.1.1	Help Topics.....	154
5.1.2	Specifications and Features:	154
5.2	Specification.....	155
5.2.1	Overview	155
5.2.2	General Specifications.....	155
5.2.3	Instruction Speed	156
5.2.4	Instruction Set.....	160
5.3	Addressing.....	163
5.3.1	Overview	163
5.3.2	Data Types.....	163
5.3.3	Data Table Addresses.....	163
5.3.4	Constants	166
5.3.5	System Control Relays.....	167
5.3.6	System Control Registers	169
5.3.7	Subroutines	170
5.4	Instructions	171
5.4.1	Overview	171
5.4.2	Instructions	171
5.4.3	Alphabetic Instruction Summary	175
5.5	Programming	181
5.5.1	Overview	181
5.5.2	PLC Programming Languages	181
5.5.3	Source File Format	181
5.5.4	Program Symbols	182
5.5.5	Ladder Representation.....	185
5.5.6	Example Program.....	187



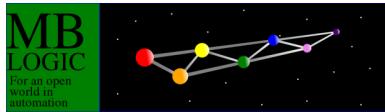
5.6	Configuration	190
5.6.1	Overview:	190
5.6.2	Configuration	190
5.6.3	System Identification	190
5.6.4	Data Table Save/Restore.....	191
5.6.5	IO Sections	191
5.6.6	Application Considerations	195
5.6.7	Complete Example	197
5.7	Program Monitoring & Reloading	201
5.7.1	Overview	201
5.7.2	Programming and Trouble Shooting.....	201
5.7.3	Starting the Soft Logic Program	201
5.7.4	Reloading the Program	201
5.7.5	Viewing Data Table Contents.....	202
5.8	Data Table Save & Reload	203
5.8.1	Overview	203
5.8.2	Data Table Save Configuration.....	203
5.8.3	Save and Restore Process	204
6	HMI.....	206
6.1	Overview	206
6.1.1	Server Configuration	206
6.1.2	Alarm and Event History	206
6.1.3	Creating HMI Web Pages	206
6.2	HMI Configuration.....	208
6.2.1	Overview	208
6.2.2	Configuration	208
6.2.3	Configuring Reserved Tags.....	208
6.2.4	Configuring Address Tags.....	209
6.2.5	Configuring Events and Alarms Tags.....	213
6.2.6	Configuring the ERP List	214
6.2.7	Examples	214



6.3	Alarm and Event History	217
6.3.1	Overview	217
6.3.2	Accessing the Alarm and Event History Page.....	217
6.3.3	Event History.....	217
6.3.4	Alarm History Query	219
6.3.5	Alarm and Event Messages.....	221
6.3.6	System Components	222
6.3.7	Customising the Alarm and Event History	223
6.4	HMIBuilder	225
6.4.1	Overview	225
6.4.2	HMIBuilder Widget Library	225
6.4.3	Inkscape Drawing Editor	225
6.4.4	MBLogic HMIBuilder	226
6.4.5	Help for HMIBuilder	227
6.5	HMI Client Libraries.....	228
6.5.1	Overview	228
6.5.2	HMI Client Protocol Library.....	229
6.5.3	HMI Client Display Library.....	241
6.5.4	HMI Client Event Library	256
6.5.5	Basic Concepts	264
7	Installation	295
7.1	Overview	295
7.1.1	Help Topics.....	295
7.2	Base Application.....	296
7.2.1	Overview	296
7.2.2	Installation	296
7.2.3	Application Directory Contents.....	296
7.3	Third Party Support Software	298
7.3.1	Overview	298
7.3.2	Obtaining and Installing Third Party Software	298
7.3.3	Installing on Mac OS/X.....	303



7.3.4	Testing Third Party Software.....	303
7.3.5	SimpleJSON	305
7.4	Starting the Application	306
7.4.1	Overview	306
7.4.2	Starting the Application	306
7.5	Testing the Application	308
7.5.1	Overview	308
7.5.2	Checking the Web Interface.....	308
7.5.3	System Help	309
7.5.4	Checking the User Help Server.....	309
7.5.5	Running the HMI Demo	309
7.6	Stopping the Application.....	312
7.6.1	Overview	312
7.6.2	Shut Down Methods	312
7.6.3	Shut Down Messages	313
7.7	Using Port 502.....	315
7.7.1	Overview:	315
7.7.2	Standard Modbus Port.....	315
7.7.3	Redirecting Port 502 on Linux.....	315
7.7.4	Port 502 on Microsoft Windows.....	317



1 Introduction



1.1 Documentation

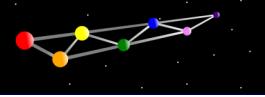
This section provides a copy of the documentation which is included in the on-line help system for the MBLogic application. The menu to the left is the menu used by the on-line help. To select a help topic from this copy of the on-line help system, choose an item from this menu. To return to the rest of the web site, choose an item from the horizontal menu above.

1.1.1 Overview

This software system provides a complete automation control system platform with the following features:

- Soft logic control using conventional PLC programming languages.
- Web based graphical HMI based on industry standards.
- Communications to field IO using industry standard client (master) and server (slave) protocols.
- No limit on the control program size or the number of communications connections.
- Web service server protocol for enterprise integration (MPR/ERP).
- Built-in web based status monitoring and control.
- Built-in web based on line help.
- Complete configurability.
- Cross platform portability.

Click on the links below or at the left to read more details on each topic.



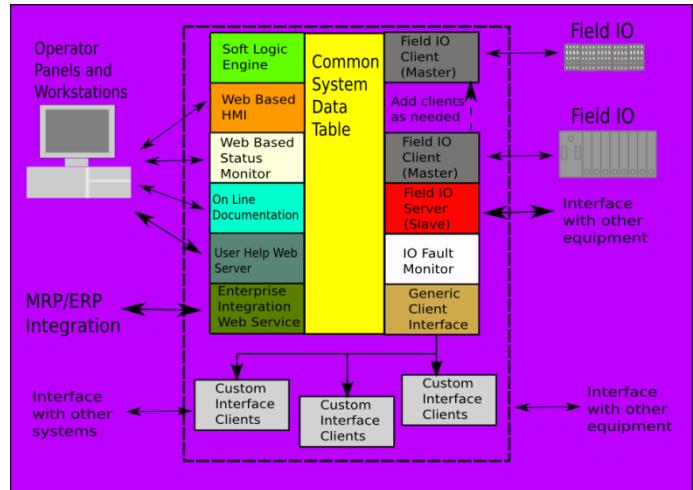
- [General](#) - General information about the packages.
- [Communications](#) - Communications and protocols, including configuration of servers and clients.
- [Soft Logic](#) - Soft logic system, including instructions, addressing, and specifications.
- [HMI](#) - Web based HMI. This includes creating web pages, and configuring the HMI server.
- [Installation](#) - System installation and start up.
- [Status](#) - How to monitor the status of a running system. Also, verifying the configuration, and making changes to a running system (on line programming).
- [About](#) - Information about the software.

2 General

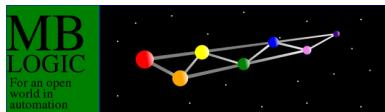
2.1.1 Overview

This system consists of a program containing a set of components, which cooperate through a shared data table (memory array). The systems include communications servers, communications clients, soft logic, and integrated on line help.

The number and type of components which are active is determined by a set of configuration files. The configuration files also control such things as protocols used, addresses, polling rates, and other factors.



- [On Line Help System](#) - Accessing on line help.
- [System Data Table](#) - The system (communications) data table.
- [Creating Applications](#) - Creating an Application.



- [File Locations](#) - Locations of System Files.
 - [System Capacity](#) - Limits of System Capacity.
 - [User Help](#) - User Created Help Pages
-

2.1.2 On Line Help

2.1.2.1 Overview

The on-line help system includes a set of web help pages which can be accessed through a built in web server. The on-line help includes information on installation, configuration, programming, and trouble shooting.

2.1.2.2 Accessing On Line Help

To access the help system, you must know the IP address of the computer where the system was installed. If your web browser is running on the same computer, you can use the address "localhost" (which is the standard address name a computer has for itself). The "home page" for the help system is called "index.html" (this is also a standard convention).

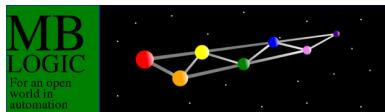
The following shows two different examples:

```
http://192.168.50.10:8080/index.html  
http://localhost:8080/index.html
```

The on-line help system is part of the web based status monitoring and control system. The port number used is set in the configuration system. The port used in the sample configuration shipped with the system is 8080.

2.1.2.3 Help System Menus

The help system is integrated into the status monitoring and control system. Each status monitoring and control web page has a link to the main help web page via a



menu option titled "Help". Click on that menu item to view the main help web page. Each help web page includes the status monitoring menu allowing you to return to the status monitoring system at any time.

Each help web page contains a series of links to help topics on the left side of the page. Click on any of these links to see more details on that topic. Below the help topic links are further links to details on the particular help topic being viewed. These detailed help links will change as different topics are viewed.

The screenshot shows a web browser window with the title "Help - On Line Help - Shiretoko". The address bar shows the URL "http://localhost:8080/general/OnLineHelp-en.html". The page content is as follows:

Help Topics

- Help
- General
- System Status
- Communications
- Soft Logic
- HMI
- Installation
- About

Topic Details for General

- On Line Help System
- System Data Table
- Creating Applications
- File Locations
- System Capacity
- User Help

Help - On Line Help

Overview:

The on-line help system includes a set of web help pages which can be accessed through a built in web server. The on-line help includes information on installation, configuration, programming, and trouble shooting.

Accessing On Line Help:

To access the help system, you must know the IP address of the computer where the system was installed. If your web browser is running on the same computer, you can use the address "localhost" (which is the standard address name a computer has for itself). The "home page" for the help system is called "index.html" (this is also a standard convention).

The following shows two different examples:

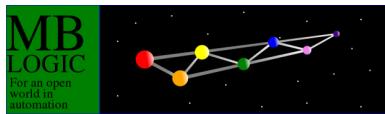
```
http://192.168.50.10:8080/index.html  
http://localhost:8080/index.html
```

The on-line help system is part of the web based status monitoring and control system. The port number used is set in the configuration system. The port used in the sample configuration shipped with the system is 8080.

Help System Menus:

2.1.2.4 What to do if the Help System is Disabled

As shipped, the help system uses port 8080 for the help web server. However, the user has the ability to assign a different port number, or to disable the web server



altogether. If the web server is disabled, the user will not be able to access the on line help (or the status monitoring system). If you wish to re-enable the web server, make sure the following lines are in the configuration file.

```
# Status web server.  
[status]  
type=tcpserver  
protocol=status  
port=8080
```

The configuration file is an ordinary text file and can be edited with any text editor. You may use a different port number if you wish. Just remember that your web browser will need to look for this alternate port number if you use it.

2.1.3 Help - Data Table

2.1.3.1 Overview

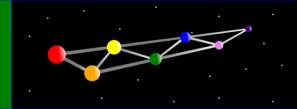
The "System Data Table" is a central array of memory which all data servers and clients and other components read or write from or to as the source or destination for each of their operations. You need to understand the system data table in order to know how to configure and use the system.

2.1.3.2 Address Range

The system data table has both bit (digital) and word (register) addresses. It is laid out according to standard Modbus convention. Alternative protocols may label the addresses differently according to their individual requirements, but the data table remains the same (note though that the soft logic system has its own independent data table). The system data table has the following address ranges:

- 65,536 coils (bits usually used for outputs).
- 65,536 discrete inputs (bits usually used for inputs).
- 1,048,576 16 bit holding registers.
- 65,536 16 bit input registers.

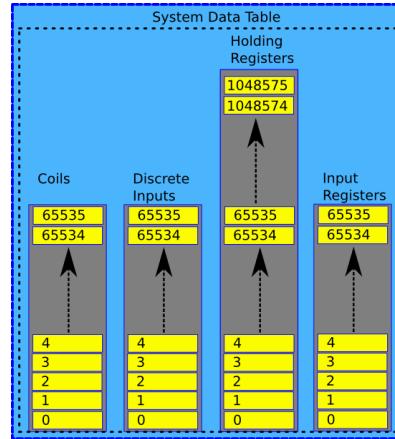
Each address type occupies a separate and independent block of memory.



2.1.3.3 Expanded Holding Register Map

Holding registers have an expanded register map. This does *not* affect the Modbus server protocol, which is still limited to a maximum address of 65,535. This is the system data table, which is the internal memory. The expanded holding register map is intended for applications which need to store large amounts of data in the data table.

Most sub-systems are able to access the expanded addresses directly as they have no inherent address limits. The Modbus/TCP server can access the expanded register addresses by using address offsets based on the unit ID. Details of this feature are discussed in the section on server configuration.



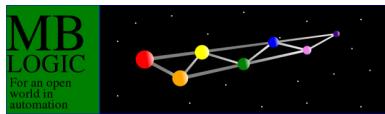
2.1.3.4 Reserved Addresses

Some addresses are reserved for special functions. This includes the fault monitoring system that reserves the upper 256 coils for resetting client faults.

2.1.3.5 Reporting Communications Faults Via the Data Table

In the communications configuration, the user can specify a set of addresses for each communications client to be used for reporting client communications errors. Each time an error is detected, these addresses are updated to indicate the error. The same error is reported all four discrete input, coil, input register, and holding register addresses. The system repeats this information to several different addresses to allow for more convenient reading of this information (the addresses can be located to be adjacent to other information which is already being read).

For discrete inputs and coils, an "off" ("0") value indicates no fault, while an "on" ("1") value indicates a fault. For input and holding registers, a "0" value indicates no



fault, while a non-zero value indicates a fault. The actual value used to indicate a fault will be protocol dependent. For Modbus, the fault number is stored in the upper byte, and the exception number is stored in the lower byte.

Fault information will be overwritten by later faults, but it will not be reset unless specifically requested.

Fault indication can be reset by writing to a special set of coil addresses. In the data table, a range of coil addresses is regularly monitored to see if any of them have been turned on. If one or more has been turned on, the system will look to see if they have been associated with a particular client fault configuration. If so, the associated fault indication addresses will be reset, and the reset coils will also be reset.

Any coils which are in the monitored range will be turned off regardless of whether or not they are associated with a fault configuration. This means that coils which are not used for fault reset may still not be used for any other purpose. Writing to a coil in the monitored range but which is not configured will not result in an error, but it will result in that coil being automatically reset.

The monitored coils are located in the upper 256 coil addresses (65279 to 65535). They are monitored once every few seconds (approximately every 2 seconds). This means that it may take up to approximately 2 seconds for a fault reset to complete once it has been requested.

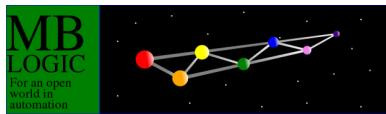
2.2 Creating an Application

2.2.1 Overview

The system can be used in a variety of applications. This section will present an example using all the components together to control a simple machine.

2.2.2 Create the Application

2.2.2.1 *Allocate the System Data Table Memory*



The communications, soft logic, and HMI systems connect with each other using the system data table before creating a new application it is necessary to allocate space in the system data table for data that needs to be read from or written to field devices, communications fault monitoring addresses, and for data that needs to be exchanged between the soft logic system and the HMI (this also goes through the system data table).

A good layout for the system data table will make configuration of the rest of the system easier. The soft logic system has its own data table and addressing system, so you can change the arrangement of data in the system data table without affecting the soft logic program or the HMI (their configurations can be adjusted to suit).

2.2.2.2 Enable the System Servers:

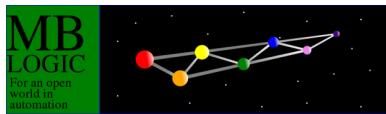
In order to be able to interact with the system, we need to enable the basic system servers. We will want the following features available to us:

- On line help. This gives us access to the on line help.
- System status and control. This gives us access to reports on the current status of the system. It also provides access to error messages for configuration problems, and it allows us to alter the configuration and soft logic program while the system is running.
- Modbus/TCP server. This enables the server (slave) protocol so that other clients (masters) can communicate with this system.
- HMI. This will allow us to create a web based HMI.
- MB-Rest web service protocol. This will provide a web service protocol which allows an MRP/ERP to communicate with our machine to obtain production data.

Information on creating a configuration is located in the section on "Communications". A sample configuration is also provided with the system.

2.2.2.3 Configure Communications with Field IO:

A normal application will require communicating with field IO such as valves, sensors, drives, etc. These field devices are normally servers ("slaves"). The system creates a separate client connection for each field device server. Each client



connection can execute multiple commands in sequence. This permits communications with each field device to take place in parallel, rather than communicating with each one at a time. This means that a slow or faulty field device will not hold up communications with other field devices.

When a client connection executes a "read" command, it will read data from a field device (remote address) and store it in the local system data table. When a client connection executes a "write" command, it will read data from the local system data table and write it to the remote field device. The HMI and soft logic systems will have access to this data through the system data table.

Information on creating a configuration is located in the section on "Communications". A sample configuration is also provided with the system.

Each field device server (slave) that you wish to communicate with requires its own client connection configuration.

2.2.2.4 Write the Soft Logic Program:

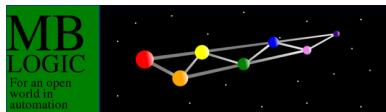
The soft logic (PLC) program determines what your machine or process will do. The system uses conventional PLC type instructions operating on a conventional data table. The soft logic data table is separate from the system data table.

See the help sections on "Soft Logic" for more information on programming, addressing, and the instruction set.

2.2.2.5 Configure the Soft Logic IO:

Since the system and soft logic data tables are separate, it is necessary to copy data between them in order for the soft logic program to have access to the field devices or HMI. This also permits the soft logic addresses to be in a different order and spacing from that used in the system data table.

The transfer of data between the system and soft logic addresses is controlled by a configuration file. The soft logic "IO" system can read and write to and from any addresses in the system and soft logic data table. See the help sections on "Soft Logic" for more information on soft logic IO configuration.



2.2.2.6 Write the HMI Web Page:

The HMI system is web based. An HMI "screen" is just a web page. See the help sections on "HMI" for more information on HMI pages.

The system is designed to communicate with multiple HMI web pages simultaneously. These can be multiple copies of the same web page, or copies of different web pages. This means that a machine or process can have multiple HMI panels if desired.

2.2.2.7 Create the HMI Server Configuration:

When the HMI client (web page) communicates with the server, it requests information in "tag label" addresses, rather than the Modbus addresses used by the system data table. The HMI server must translate between the "tag" names used by the HMI client and the numerical addresses used by the system data table. For example, "PB1" may be stored in coil address "100", and "PB2" may be stored in coil address 25. Tag labels are associated with system data table addresses by means of a configuration file. See the help sections on "HMI" for more information on HMI configuration.

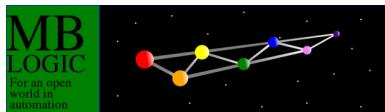
2.2.2.8 Load the Configurations and Programs:

Once the configurations, program, and web page(s) have been created, they need to be loaded into the system. The communications configuration can only be changed by stopping and restarting the entire system. The other (soft logic configuration and program, HMI configuration, HMI web pages) can (except for a few items) be reloaded while the system is running.

2.2.2.9 Debugging and Troubleshooting:

The current configurations can be viewed using the status system web interface. The status system also allows any address (or set of addresses) to be inspected while the system is running. For more information on using the status system consult the on line help pages included with the status system.

2.3 File Locations



2.3.1 Overview

This help topic describes the names and locations of important system files.

2.3.2 System Files Location

The system program files are located in a subdirectory called "mblogic". The system expects this sub-directory to be located in (branching off from) the "current directory" (the directory from which the application was started). If the directory is *not* located in the expected place, the web servers which provide the on-line help and status information will not be able to find the directories where the web pages are stored.

When you are starting the program, make sure that you are in the correct directory before starting the program. If you start it using the provided shell script or batch file, this should not be a problem. If you are creating your own start up method, be sure to change into the correct directory before starting the program.

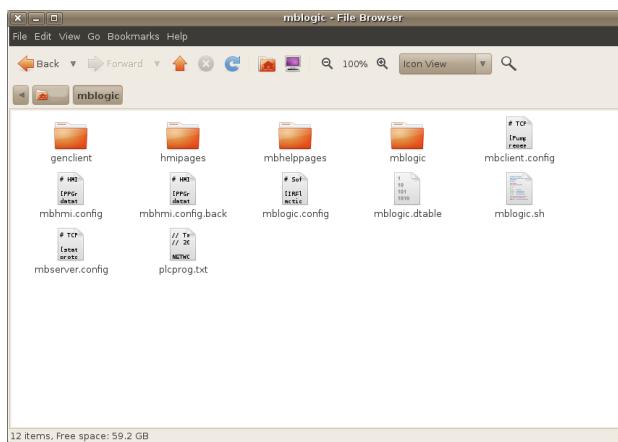
2.3.3 Configuration Files and Soft Logic Program

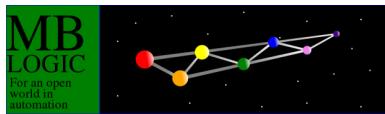
The system expects to find the configuration files and the soft logic program in the "current directory".

2.3.3.1 Application File Locations

These files are:

- **mbserver.config** - This configures the server communications connections, including the on-line help server and the status system server. The system will not function without a correct configuration.
- **mbclient.config** - This configures





the client communications connections.

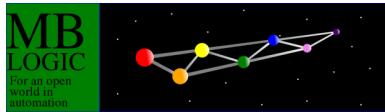
- **mbhmi.config** - This configures the server functions for the HMI. The HMI system will not function without a correct configuration. The communications port number used by the HMI server however, is set in "mbserver.config".
- **mblogic.config** - This configures the exchange of data between the soft logic system and the other components in the system. The soft logic system will not function without a correct configuration.
- **Soft logic program** - The name of the file used for the soft logic program is user configurable via the mblogic.config file. There is no default name for this file. The system ships with a sample program called "plcprog.txt". However, any file name and any file extension can be used, provided it does not conflict with the name used by one of the other expected files. For example, "MACH021.plcpgr" is a valid name.
- **mblogic.dtable** - This is a system file which is used to store data table values. This permits data table values to be restored after a system restart. Details on this feature may be found in the help section on "Soft Logic". This file is automatically created on start up and the size of the file depends on the soft logic configuration.

2.3.3.2 Backup Files

When configuration files are edited via the configuration forms, the old versions are saved with a ".back" extension. For example, "mbhmi.config" will be saved as "mbhmi.config.back".

2.3.4 HMI Files

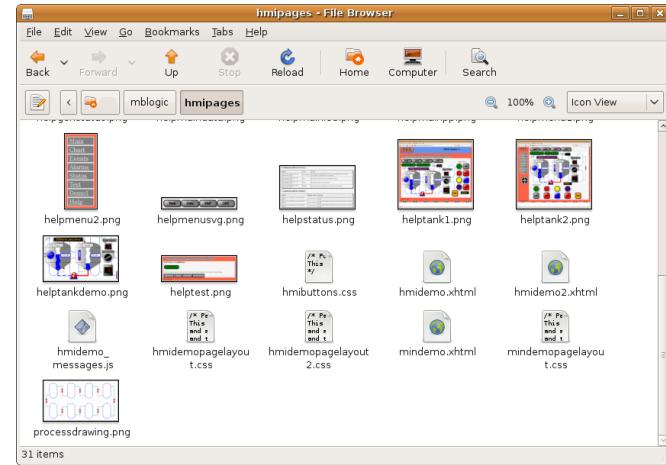
The system expects to find the HMI client files in a directory called **hmipages** branching off from the "current directory". The HMI client web page can have any name. The HMI web server will fetch any page in that directory requested by the user. This includes any Javascript, CSS, PNG, and other files which are named in the web page.



2.3.4.1 HMI File Locations

Since the HMI web server will serve any web pages, user generated web pages for purposes other than HMI can also be placed in this directory. For example, equipment manuals and operator instructions can also be served from this directory. However, a dedicated web server for user documentation is also provided (see below).

The standard Javascript library files are located in the application directory in a subdirectory called "hmilib". The HMI system will automatically search for them there.



2.3.5 User Help Pages

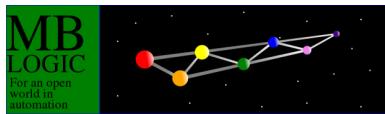
The system includes a simple web server that can be used to host user created web pages for purposes such as equipment manuals, operating procedures, etc. The system expects to find these files in a directory called ***mbhelppages*** branching off from the "current directory". The web pages themselves can have any name. The help web server will fetch any page in that directory requested by the user. This includes any Javascript, CSS, PNG, and other files which are referenced by the web pages.

2.3.6 Generic Clients

Generic clients are stored in the ***genclient*** directory.

2.4 System Capacity

2.4.1 Overview



There is no limit on the number of communications connections or the size of the soft logic program permitted. However, each client or server connection requires CPU time in which to execute. Excessive of communications or soft logic program load can load down the PC's CPU, leaving no time for other processes to execute.

2.4.2 Maximum System Capacity

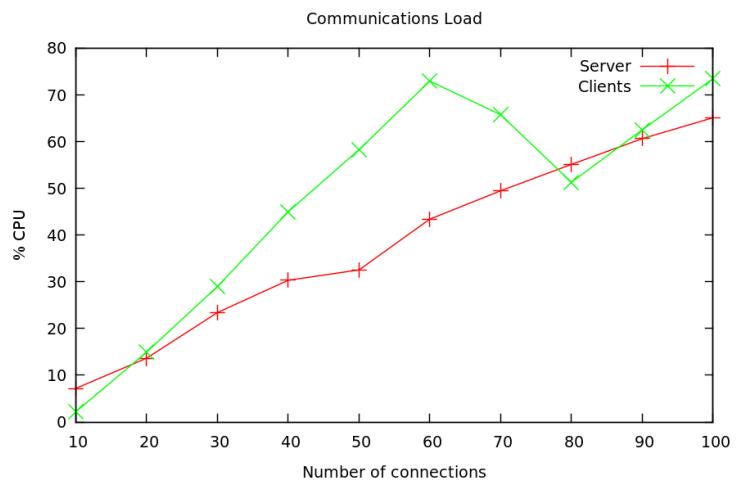
The following tests were conducted on a relatively slow PC. A faster PC would permit more connections or a larger program, or a faster polling rate. These samples should be used only as a general indication, and you should observe the readings with your own application on your own hardware to determine actual system load.

2.4.2.1 Effect of Client and Server Connections on CPU Load:

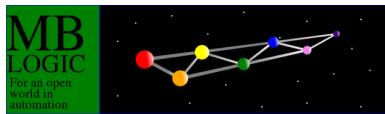
All connections were made over Ethernet (not to localhost). The server test involved independent clients reading 128 coils at a time (function 1, quantity 128), at 50 millisecond intervals.

The client test involved making multiple connections to a server and reading 128 coils at a time (function 1, quantity 128), at 50 millisecond intervals.

Other tests (not shown here) indicate that system load is mainly determined by the number of messages per second. The size of the individual messages is much less important. Reading or writing one large block of data is much more efficient than several smaller reads.



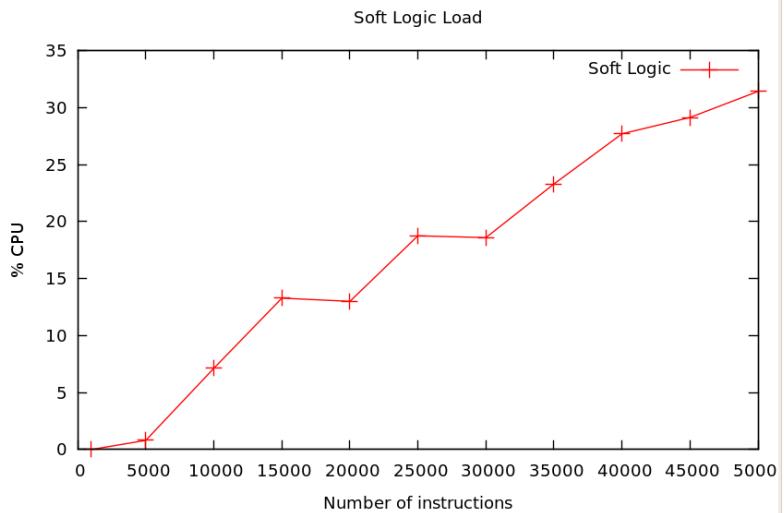
2.4.2.2 Effect of Program Size on CPU Load:



The test involved executing boolean instructions at a target scan delay of 50 milliseconds with an instruction mix of:

- 10% STR (push a value onto the logic stack)
- 50% AND
- 20% OR
- 10% OUT
- 10% SET

Note that the soft logic system now operates on the principle of set delays between scans, not fixed scan rates. Actual scan rate will be the result of the sum of the scan delay target plus the time to actually execute the scan.



2.4.2.3 Effect of HMI Pages on CPU Load:

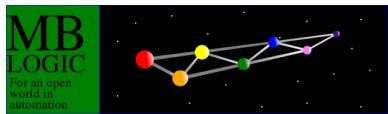
The HMI client (web page) imposes approximately 3% average CPU load when running the HMI demo included with the system (when the HMI is idle). The graphics used in the demo are (deliberately) very simple. SVG is capable of elaborate special effects which requires a lot of hardware processing. If you decide to make use of some of these special effects, you may wish to check the effect on CPU load and possibly install a good graphics board.

2.5 User Created Help Pages

2.5.1 Overview

The system includes a built-in web server that can be used to serve user created web pages. These web pages can be used to display equipment manuals, procedures, or any other content desired.

2.5.2 Web Pages



The system includes a web server for simple *static* web pages. A static web page is one whose content doesn't change (the status system by contrast uses dynamic web pages to update them with new content).

The web pages can include text, graphics, CSS, javascript, and any other normal web page components.

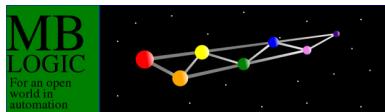
2.5.3 Web Site Configuration

Creating a user web site simply requires copying the user created web pages into the help page directory. The port number used by the web server can be configured via the communications configuration (see the help section on "Communications" for more details). The web server is configured as a "help" server (protocol=help).

2.5.4 Web Site File Location

The web site files are located in the "mbhelppages" directory. Anything placed in this directory becomes part of the web site. This may include subdirectories.

The system ships with a default place holder web page, including graphics and CSS. These files are not necessary for system operation and may be removed or replaced with user created files.



3 Status Status

3.1 Overview

The "Status" system is a web based interface which allows you to access information about the current configuration and run time status of this system.

This section covers the "Status" web based interface. For more general help about installation, configuration, programming, and other subjects, see the other help topics.

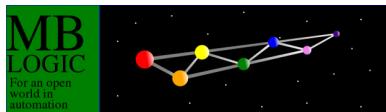
To access a particular help topic, click on one of the links in the list below. The "Status" system consists of the following pages:

- [System Status Summary](#) A quick overview of the current system status.
- [Configure Summary](#) An overview of the current configuration.
- [Editing Configurations](#) Editing and saving configurations.
- [Servers Configuration](#) Display of the current server communications configuration.
- [Clients Configuration](#) Display of the current client communications configuration.
- [HMI Configuration](#) Display of the current HMI configuration.
- [Soft Logic IO Configuration](#) Display of the current soft logic IO configuration.
- [Soft Logic Program](#) Display of the soft logic program. Also add, edit, delete the soft logic program.
- [Control](#) Control reloading configurations and programs from disk.
- [Monitor Data](#) Monitor the soft logic program and the soft logic and system data tables.
- [Monitor System](#) Monitor the system operation.
- [Cross Reference](#) Soft logic program cross references.

3.2 System Status

3.2.1 Overview

The "System Status" page provides you with a quick overview of the current status of the software system. This page will automatically continuously update itself with the current status of the system.



The screenshot shows a web browser window titled "System Status - Minefield". The address bar indicates the URL is <http://localhost:8080/statussystem.html>. The page header features the MBLogic logo and the tagline "for an open world in automation". Below the header is a navigation menu with links: Home, Configure, Control, Monitor Data, System, XRef, and Help. The main content area is divided into two sections: "General System Parameters" and "Subsystem Status".

General System Parameters

System Name	Loopback Server
Software	MBLogic
Version	01-Sep-2010
Started at	Tue 31 Aug 2010 03:19:20 PM EST
Uptime (hrs)	0.00

Subsystem Status

Servers	6
TCP Clients	8
Generic Clients	1
HMI	Ok
Soft Logic IO	Ok
Soft Logic	Running
Status Client	Ok

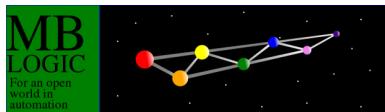
3.2.2 General System Parameters

The General System Parameters displays the name and version of the application software, and also how long the system has been running since its last restart.

- **System Name** - This is the name of the system as set in the system configuration file.
- **Software** - This is the name of this application software package. This cannot be changed except by editing the source code and
- **Version** - This is the version number of the application software.
- **Started at** - This is the date and time at which the system was started.
- **Uptime (hrs)** - This is the time (in hours) since the system was last started.

3.2.3 Subsystem Status

Subsystem Status shows the current status of each of the major subsystems. As well as a textual indication, each of the subsystems is colour coded for quick reference. The colour codes used can be changed by editing the web page CSS, but the default values used are:



3.2.3.1 Colour Codes

- Green - OK - No problems are evident and everything appears to be running normally.
- Yellow - Alert - An error may have occurred when attempting to load a new version of a configuration or soft logic program. This may result in a fault if the system is restarted.
- Red - Fault - A major problem has been detected and should be corrected.

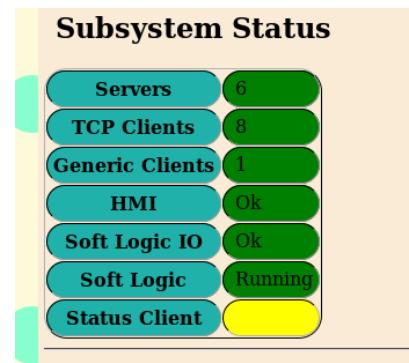
3.2.3.2 Monitored Subsystems

The following subsystems are being monitored.

- Servers - The number of communications servers active.
- Clients - The status of all of the communications clients.
- Soft Logic - The status of the soft logic system.
- Soft Logic IO - Whether any problems were detected in the soft logic IO configuration.
- HMI - Whether any problems were detected in the HMI configuration.

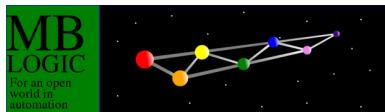
3.2.3.3 Browser Watchdog

The status page runs a watchdog timer which tracks whether it is able to contact the main system status server. The result of this is displayed as "Status Client". When the status client (web browser) is able to poll the status server, the result will be displayed as OK (green, under the default the colour configuration). If the watchdog timer has timed out, the result will be displayed as "alert" (yellow, under the default the colour configuration).



3.2.4 Static Web Status Page

Some web browsers are based on older internet technologies and are not capable of all the features used by the status monitoring system. Microsoft Internet Explorer is the



best known of these older browsers. When a web browser asks for the system status page, the system looks at the request headers to see what type of web browser is being used. If Microsoft Internet Explorer (any version) is detected, the system automatically redirects it to a special status web page which uses only static data. This static web page does not update automatically. You will need to reload the page (press "F5") to update the page with new data. The only menu options available will be the summary page ("Home") and the help pages.

Web browsers which are based on modern standards will work with all features of the status interface. This includes virtually all other web browsers other than Microsoft Internet Explorer. Firefox, Google Chrome, Opera, Apple Safari, Epiphany, and most others should work with all features. Using any of these modern web browsers is strongly recommended.

3.3 Configure Summary

3.3.1 Overview

The "Configure Summary" page provides with a quick overview of the state of the running configuration of each of the sub-systems.

3.3.2 Configurations

The available system configurations are:

- Servers
- Clients
- HMI
- Soft logic IO
- Soft logic program

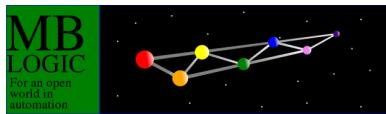
The sub-systems are displayed in a table with the following items for each.

The screenshot shows a web browser window titled "System Configure - Minefield". The address bar shows the URL <http://localhost:8080/statuspages/statusconfigure.html>. The page header includes the MB Logic logo and the slogan "for an open world in automation". The main menu bar has links for Home, Configure, Control, Monitor Data, System, XRef, and Help. The "Configure" section displays a table of currently running configurations:

Sub-System	Status	Started At	Signature
Servers	Ok	Tue 19 Oct 2010 12:21:42 AM EST	766dd48d55686d33bf9cfa55f6fb37
Clients	Ok	Tue 19 Oct 2010 12:21:42 AM EST	c4fad248e403589f80bd20ac8b8b2c8a
HMI	Ok	Tue 19 Oct 2010 12:21:43 AM EST	4f61bf27b32a324a7cc71ada3cf048b
Soft Logic IO	Ok	Tue 19 Oct 2010 12:21:43 AM EST	24e6c9f7aba78d5653fa1088838cf6a7
Soft Logic Program	Ok	Tue 19 Oct 2010 12:21:43 AM EST	f9a6df02b6542afa145d0af1abbca18a

The "HMI Web Pages" section shows two entries:

HMI Web Page	Signature
hmidemo.xhtml	0826c99d05785fd5b2560e3bf5acee28
hmidemo2.xhtml	a430f978fcac07c034c37ac351509bbb

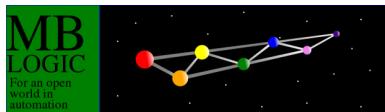


- Sub-system name. Click on the name to open a page with more details.
 - Status. This contains a text and colour coded background to indicate the current status of each sub-system.
 - Started at. This is the time the sub-system was started.
 - Signature. This is an MD5 checksum of the configuration. This checksum may be recorded and compared later to determine if any changes have been made to the configuration files.
-

3.3.3 HMI Web Pages

Also displayed are any web pages contained in the HMI web page directory. This shows the pages that were present when the configure web page was requested. To update the list, refresh the configure web page. The HMI web page information includes:

- The web page file name. Click on the name to open the HMI web page.
- Signature. This is an MD5 checksum of the web page. This checksum may be recorded and compared later to determine if any changes have been made to the HMI web page files.



3.4 Edit Configurations

3.4.1 Overview

This section describes how to use the form based editors to create and change system configurations. This allows editing of the following:

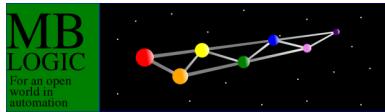
- Servers
- Clients
- HMI
- Soft Logic IO

Soft logic programs are not discussed here, but are covered elsewhere in the documentation.

3.4.2 Enabling Editing

When a configuration web page is displayed, it is in "view only" mode. This allows you to view the configuration settings, but not change them. To place the configuration page into edit mode, you select "edit" from the radio button widget near the upper left corner of the page (below the menu bar).





Home Configure Control

Soft Logic IO Configuration

View Only: Edit:

When the page is in edit mode, items which can be edited will change colour as you move your mouse cursor over them. This shows you which items can be edited and which cannot.

Server protocol	Server Name	IP Port
modbustcp	Main1	8502
hmi	HMI	8082
status	status	8080
help	WebHelp	8081
generic	Generic	8000

3.4.3 Input Forms

Input is entered via forms which become visible when you select an item to enter. The forms contain text and numeric entry boxes, radio buttons, drop down selection lists, etc.

Home Configure Control Monitor Data System XRef Help Save All

Communications Client Configuration

View Only: Edit:

TCP Clients

Connection Name	Protocol	IP Address
PumpPressure	modbustcp	localhost
FanCommand	modbustcp	localhost
Robot1	modbustcp	localhost
FanSpeed	modbustcp	localhost
Beacon	modbustcp	localhost
SlidePosition	modbustcp	localhost
Conveyors	modbustcp	localhost
Horn	modbustcp	localhost

TPC Client Parameters

Connections: Polling: Faults: Commands:

Command:

Function Code:

Remote Address:

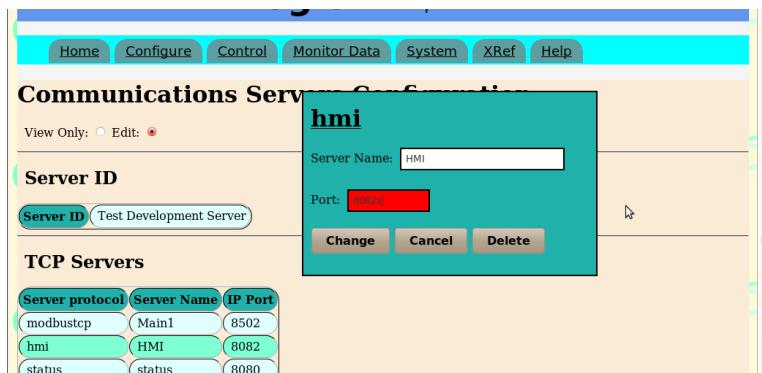
Quantity:

Add or delete connection:

Buttons: Change, Cancel, Delete

3.4.4 Validating Input

Most inputs are validated immediately when you enter them. An invalid entry in an input fields will cause the background to change colour indicating an error. You will not be able to save the data in the form until all errors are corrected. The validity of data in some fields may depend on the value of the data in other fields.



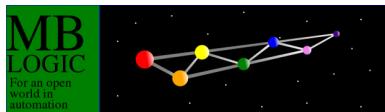
3.4.5 Saving Data

When you set the web page to "edit" mode, a "save all" button will appear in the upper right corner of the page. This will move if you scroll the page so as to remain always visible.

When you enter data in a form, it is *not* saved to disk until you press the "save all" button. When you press the "save all" button, it sends the data to the server and requests that it be validated and saved to disk. The "save all" button is disabled while you are editing a form.



When data is sent to the server to be saved a message is briefly displayed to indicate this is taking place. The message is displayed while waiting for the server to acknowledge the save operation, plus a short additional time delay (the time delay was



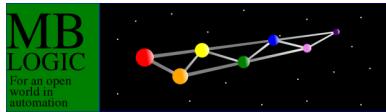
added because without it a save operation would normally happen too quickly for an operator to see the acknowledgement).



3.4.6 Server Validation

When data is sent to the server to be saved, the server will run the same series of checks on it that it uses to validate the data when loading it from disk. The server may find errors that were not detected by the web browser form (e.g. the server knows more about correct address formats).

If an error is found, the data will *not* be saved. Instead, an error message (or series of messages) will be returned to the web browser indicating what the error was. These errors must be corrected before a save operation can be completed.



Soft Logic IO Configuration Details

View Only: Edit:

Errors Saving Configuration:

1 Soft logic IO config error in Alarms - unknown address ['BADDADDRESS']
2 Soft logic IO config error in Alarms - unknown address ['Y30', 'Y31', 'Y32', 'Y33', 'Y34', 'Y35', 'Y36', 'Y37', 'Y38', 'BADDADDRESS']

Soft Logic System Parameters

Soft Logic Engine Type: ck

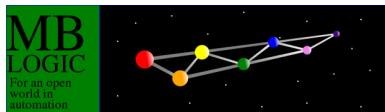
Processor File: [Processor File](#)

Save All

3.4.7 When Changes Take Effect

Changes to data will not take effect until they are successfully saved to disk. Changes to the HMI or soft logic IO configuration take effect immediately upon being saved.

Changes to the client or server communications on the other hand do not take effect until the system has undergone a complete restart. If you make a change to the client or server communications but do not perform a system restart, the running configuration will not be the same as the configuration stored in the file(s) on disk. If you reload the display form, the *current* configuration will be shown which may not reflect those changes. When you make a change to the client or server configuration, you should follow it with a system restart.



3.5 Configure Server Communications

3.5.1 Overview

The "Configure Servers" page provides an overview the current configuration of the server communications subsystem. This is divided into the following sections:

- Servers ID
 - TCP Servers
 - Data Table Expanded Register Map
-

3.5.2 Server ID

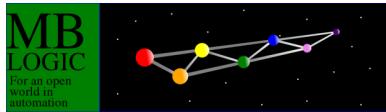
This section shows the server id configuration. The server id is the "name" given to the system to identify it (e.g. "Press1").



3.5.2.1 Editing the Server ID

To edit the server id, enable configuration editing and then click on the server id. Edit the server id string and then press "change".





3.5.3 Servers

The screenshot shows a web-based configuration tool for communications servers. At the top, there's a navigation bar with links for Home, Configure, Control, Monitor Data, System, XRef, and Help. Below this is a main title "Communications Servers Configuration". A "View Only" button is present. The main content area is divided into sections: "Server ID" (containing a "Server ID" field set to "Test Development Server"), "TCP Servers" (a table showing server protocols, names, and IP ports), and "Data Table Expanded Register Map" (with tabs for "Expanded Register Map" and "Incremental"). The "TCP Servers" table data is as follows:

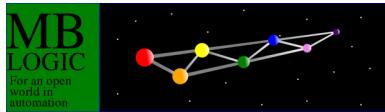
Server protocol	Server Name	IP Port
modbustcp	Main1	8502
hmi	HMI	8082
status	status	8080
help	WebHelp	8081
generic	Generic	8090
mbrest	Web1	8085

The servers table lists the active servers. The items of data include:

- The name of the server as assigned in the configuration file.
- The server protocol.
- The IP port used.

3.5.3.1 Editing the Server Data

To edit the server data, enable configuration editing and then click on the server you wish to edit. Edit the server name or port number and then press "change". You may delete the server by pressing the "delete button". Deleting a server does not remove the row from the table. Rather it simply clears the information from that server type. Since there may be only one of each type of server protocol present, all available server types are displayed whether they are configured (and active) or not.



hmi

Server Name:

Port:

Change **Cancel** **Delete**

3.5.4 Data Table Expanded Register Map

Communications Servers Configuration - Minefield

TCP Servers

Server protocol	Server Name	IP Port
modbustcp	Main1	8502
hmi	HMI	8082
status	status	8080
help	WebHelp	8081
generic	Generic	8090
mbrest	Web1	8085

Data Table Expanded Register Map

(Expanded Register Map) (Incremental)

Show UID Details

Hide Details

Unit ID	Offset
1	0
3	131072
5	262144
7	393216
9	524288
11	655360
13	786432
15	917504

Unit ID	Offset
2	65536
4	196608
6	327680
8	458752
10	589824
12	720896
14	851968
16	983040

Communications Configuration

This lists the data table expanded register map values used. This includes:

- The mode (disabled or incremental).
- A table listing the configured unit IDs and the resulting calculated register map offsets.

3.5.4.1 Editing the Register Map

Expanded Register Map Configure

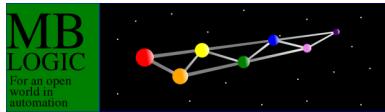
Expanded Register Map: Disable Enable

Unit ID Offset:

Register Factor:

Change

Cancel



To edit the expanded register map parameters, enable configuration editing and then click on the expanded register map configuration. Edit the parameters and then press "change". If you select "disable", the uid offset and register factor are not used.

3.6 Configure Client Communications

3.6.1 Overview

The "Configure Clients" page provides an overview the current client communications configuration subsystem. This is divided into the following sections:

- TCP Clients
- Generic Clients

Communications Clients Configuration

View Only: Edit:

TCP Clients

Connection Name	Protocol	IP Address	IP Port	Cmd Action	View Details
PumpPressure	modbustcp	localhost	8502	poll	<button>View</button>
FanCommand	modbustcp	localhost	8502	poll	<button>View</button>
Robot1	modbustcp	localhost	8502	poll	<button>View</button>
FanSpeed	modbustcp	localhost	8502	poll	<button>View</button>
Beacon	modbustcp	localhost	8502	poll	<button>View</button>
SlidePosition	modbustcp	localhost	8502	poll	<button>View</button>
Conveyors	modbustcp	localhost	8502	poll	<button>View</button>
Horn	modbustcp	localhost	8502	poll	<button>View</button>

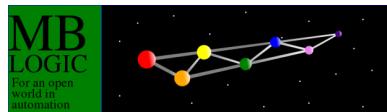
Generic Clients

Connection Name	Protocol	Cmd Action	View Details
HartClient	hart	poll	<button>View</button>
MBRTUClient	modbusrtu	poll	<button>View</button>

Communications Configuration

3.6.2 TCP Clients

The TCP clients table lists the active TCP clients. The items of data include:



- The connection name as assigned in the configuration file.
- The protocol
- The IP address to connect to.
- The IP port to connect to.
- The command action.
- A button which may be clicked to view more details.

When more details are selected, a new area will open up just below the client table to show the details for one client at a time. Selecting another details button will show the details for a different server. Selecting "hide details" will hide the details section of the page.

Communications Clients Configuration - Minefield

Client Name	Protocol	IP Address	Port	Action
FanSpeed	modbustcp	localhost	8502	poll
Beacon	modbustcp	localhost	8502	poll
SlidePosition	modbustcp	localhost	8502	poll
Conveyors	modbustcp	localhost	8502	poll
Horn	modbustcp	localhost	8502	poll

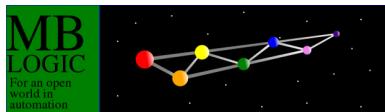
Client Details for: Conveyors

Polling Times		Fault Addresses	
Item	Time (msec)	Item	Address
Cmd Time	50	Coil	10001
Repeat Time	50	Discrete Input	10001
Retry Time	5000	Holding Register	10001
		Input Register	10001
		Reset Coil	65280

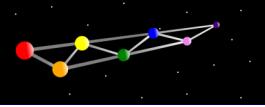
Command List

Command Name	Function Code	Remote Memory Address	Quantity	Local Memory Address	Unit ID
readcoils	1	1	1	1001	2
readcoils2	1	2	8	1008	1
readcoils3	1	16	12	1016	1
readcoils4	1	100	100	1100	1
readcoils5	1	10000	123	10000	1

The correspondence between the configuration parameters and the web page labels is shown below.



Item	Configuration Label	Description
Connection Name	[section heading]	The name of the client connection.
Protocol	protocol	The type of client protocol.
IP Address	host	The target IP address.
IP Port	port	The target IP port.
Cmd Action	action	The command action requested.
Cmd Time (msec)	cmdtime	The time delay (in milliseconds) between commands.
Repeat Time (msec)	repeattime	The time delay (in milliseconds) before all commands are repeated.
Retry Time (msec)	retrytime	The time delay (in milliseconds) before retrying after an error.
Discrete Input	fault_inp	The discrete input to use for reporting faults.
Coil	fault_coil	The coil to use for reporting faults.
Input Register	fault_inpreg	The input register to use for reporting faults.
Holding Register	fault_holdingreg	The holding register to use for reporting faults.



Reset Coil	fault_reset	The coil to use to reset the fault.
------------	-------------	-------------------------------------

3.6.2.1 Modbus Commands

Modbus commands have the following parameters:

- Function code - The Modbus function code.
- Remote Memory Address - The memory address on the field device.
- Quantity - The number of elements to read or write.
- Local Memory Address - The memory address in the system data table.
- Unit ID - The unit id sent to the field device.

3.6.2.2 Editing TCP Clients

To edit a TCP client, enable configuration editing and then click on a client in the display table. Edit the parameters and then press "change". To delete a client, press the "delete" button. To add a new client, press the "Add" button at the bottom of the table.

The client editing form is divided into the following sections:

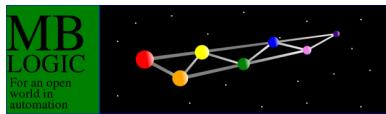
- Connection
- Polling
- Faults
- Commands

Click on the appropriate selection to edit those parameters.

3.6.2.2.1 Connection

This defines the connection parameters including:

- Connection name - This can be any valid name.
- IP Address - The TCP/IP address to connect to.
- IP Port - The TCP/IP port to connect to.



TPC Client Parameters

Connections: ● Polling: ○ Faults: ● Commands: ●

Connection Name:

IP Address: IP Port:

Change **Cancel** **Delete**

3.6.2.2.2 Polling

The polling parameters include the following:

- Command time.
- Repeat time.
- Retry time.

All times are in milliseconds.

TPC Client Parameters

Connections: ● Polling: ○ Faults: ● Commands: ●

Polling Times:

Command: Repeat: Retry:

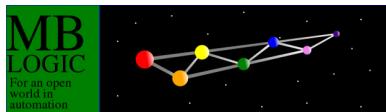
Change **Cancel** **Delete**

3.6.2.2.3 Faults

The fault parameters are the data table addresses in which to report communications errors. These include the following:

- Coil
- Discrete Input
- Holding Register
- Input Register
- Reset Coil

All addresses must be valid data table addresses.



TPC Client Parameters

Connections: Polling: Faults: Commands:

Fault Addresses:

Coil: Discrete Input:

Holding Reg: Input Reg:

Reset Coil:

Action Buttons:

3.6.2.2.4 Commands

The command parameters are the client communications commands which are to be executed. These include the following:

- Command - The command name. If you have not added any commands there will be nothing you can select here.
- Function Code - The Modbus function code.
- Remote Address - The data table address in the field device.
- Local Address - The data table address in the local system.
- Quantity - The number of data table addresses to read or write. The valid range for this will vary depending on the function code selected.
- Unit ID - The Modbus unit id to send with the client request. Whether or not the value is significant depends on whether or not the field device being addressed uses it.

TPC Client Parameters

Connections: Polling: Faults: Commands:

Command: ▾

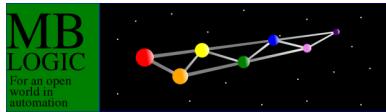
Function Code: ▾

Remote Address: Local Address:

Quantity: UnitID:

Add or delete command:

Action Buttons:



To add a new command, press the "add" button. This will open a new dialogue which allows a new command name to be entered. To delete a command select the command you wish to delete and then press the "delete" button (beside the "add" button).

TPC Client Parameters

Connections: • Polling: • Faults: • Commands: •

New Command Name:

Buttons: Change | Cancel | **Add** | **Delete**

3.6.3 Generic Clients

Communications Clients Configuration - Minefield

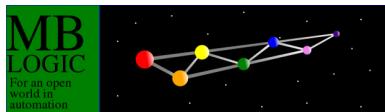
Client Details for: HartClient

Client Start Parameters		Fault Addresses	
Item	Parameter	Item	Address
Client File	hartclient.py -d 1	Coil	12500
Restart on Fail	yes	Discrete Input	12500
		Holding Register	12500
		Input Register	12500
		Reset Coil	65297

Data Table Read Addresses			Client Parameters		
Item	Address	Length	Item	Parameter	
Coil	0	0	Repeat Time	1000	
Discrete Input	0	0	Retries	0	
Holding Register	0	0	Serial Port	/dev/ttys1	
Input Register	0	0	statisticstable	60	

Data Table Write Addresses		
Item	Address	Length
Coil	11500	20
Discrete Input	11500	125
Holding Register	11500	25
Input Register	11500	100

Command List	
Command Name	Command Parameters
&readprimaryvar	action=poll, uid=0, function=1, datatype=inpreg, dataoffset=0
&readuniqueid	action=poll, uid=0, function=0, datatype=holdingreg, dataoffset=10

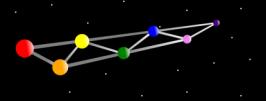


The generic clients table lists the active generic clients. The items of data include:

- The connection name as assigned in the configuration file.
- The protocol
- The command action.
- A button which may be clicked to view more details.

When more details are selected, a new area will open up just below the client table to show the details for one client at a time. Selecting another details button will show the details for a different server. Selecting "hide details" will hide the details section of the page.

Item	Configuration Label	Description
Connection Name	[section heading]	The name of the client connection.
Protocol	protocol	The type of client protocol.
Cmd Action	action	The command action requested.
Client File	clientfile	The client file name.
Restart on Fail	restartonfail	The restart on fail characteristics.
Data Table Read Addresses	readtable	The addresses to read from the main system and copy to the generic client. Unused addresses are displayed as blanks.
Data Table Write Addresses	writetable	The addresses to read from the generic client and copy to the main system. Unused addresses are displayed as blanks.



(Fault) Discrete Input	fault_inp	The discrete input to use for reporting faults.
(Fault) Coil	fault_coil	The coil to use for reporting faults.
(Fault) Input Register	fault_inpreg	The input register to use for reporting faults.
(Fault) Holding Register	fault_holdingreg	The holding register to use for reporting faults.
(Fault) Reset Coil	fault_reset	The coil to use to reset the fault.

3.6.3.1 Commands

Commands are parsed into the command name and the command parameters. The command parameters are displayed as a single block.

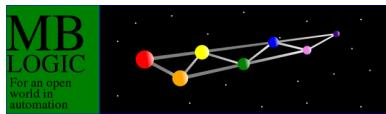
3.6.3.2 Editing Generic Clients

To edit a generic client, enable configuration editing and then click on a client in the display table. Edit the parameters and then press "change". To delete a client, press the "delete" button. To add a new client, press the "Add" button at the bottom of the table.

The client editing form is divided into the following sections:

- Client Start
- Data Table Read
- Data Table Write
- Faults
- Client Parameters
- Command List

Click on the appropriate selection to edit those parameters.



3.6.3.2.1 Client Start

The client start parameters include the following:

- Connection Name
- Protocol
- Client File
- Restart on Fail

Generic Client Parameters

Client Start: Data Table Read: Data Table Write: Faults: Client Params: Command List:

Connection Name: HartClient Protocol: hart

Client File: hartclient.py -d1 Restart on Fail: Do Not Restart: Do Not Start:

Change Cancel Delete

3.6.3.2.2 Data Table Read

The addresses to read from the main system and copy to the generic client.

Generic Client Parameters

Client Start: Data Table Read: Data Table Write: Faults: Client Params: Command List:

Data Table Read Addresses:

Coil: 0 Length: 0 Discrete input: 0 Length: 0
Holding Register: 0 Length: 0 Input Register: 0 Length: 0

Change Cancel Delete

3.6.3.2.3 Data Table Write

The addresses to read from the generic client and copy to the main system.

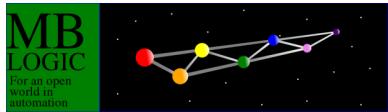
Generic Client Parameters

Client Start: Data Table Read: Data Table Write: Faults: Client Params: Command List:

Data Table Write Addresses:

Coil: 11500 Length: 20 Discrete input: 11500 Length: 125
Holding Register: 11500 Length: 25 Input Register: 11500 Length: 100

Change Cancel Delete



3.6.3.2.4 Faults

The addresses used to report errors.

Generic Client Parameters

Client Start: Data Table Read: Data Table Write: Faults: Client Params: Command List:

Fault Addresses

Coil: Discrete Input: Holding Reg: Input Reg:
Reset Coil:

Action Buttons: Change, Cancel, Delete

3.6.3.2.5 Client Parameters

Client parameters are edited via a simple text box. Since each generic client can define its own parameters they cannot be verified by the form. Verification must be done by the generic client itself at run time. Client parameters are added to the configuration file exactly as written.

Generic Client Parameters

Client Start: Data Table Read: Data Table Write: Faults: Client Params: Command List:

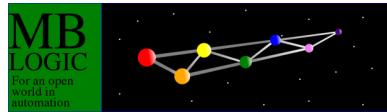
Client Parameters

```
retries=1000
retries=0
serialport=/dev/ttyS1
serialbaudrate=60
serialbytesize=100
serialparity=none
serialstopbits=1
```

Action Buttons: Change, Cancel, Delete

3.6.3.2.6 Command List

Commands are edited via a simple text box. Since each generic client can define its own commands they cannot be verified by the form. Verification must be done by the generic client itself at run time. Commands are added to the configuration file exactly as written.



Generic Client Parameters

Client Start: ● Data Table Read: ● Data Table Write: ● Faults: ● Client Params: ● Command List: ○

Commands

```
6c3ad01primaryaction=poll, uid=0, function=1, datatype=integer, dataoffset=0
6c3ad01uid=action=poll, uid=0, function=0, datatype=holdingreg, dataoffset=10
6c3ad01var=action=napshot, uid=0, function=3, datatype=integer, dataoffset=10
6c3ad01var=action=holdreg, uid=0, function=2, datatype=holdingreg, dataoffset=0
```

Buttons: Change, Cancel, Delete

3.7 Configure HMI

3.7.1 Overview

The "Configure HMI" page provides an overview the current configuration of the HMI subsystem. This is divided into the following sections:

- HMI ID
 - Tag configuration
 - Alarm and event tables
-

HMI Details - Minefield

File Edit View History Bookmarks Tools Help

http://localhost:8080/statuspages/statushmi.html

HMI Details

MBLogic for an open world in automation

Home Configure Control Monitor Data System XRef Help

HMI Configuration Details

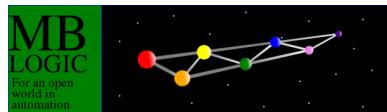
View Only: ● Edit: ●

HMI ID

Server ID	MBLogic demo server
Client Version	Ver 0.91 Demo

Address Tag Configuration

Tag Name	Address Type	Data Type	Memory Address	Minimum Range	Maximum Range	Scale Offset	Scale Span
ButtonDisPB	coil	boolean	32204				
ButtonDisSS	coil	boolean	32203				
Float32Demo	holdingregfloat	float	32217	-2147483648	2147483647	0	1
Float64Demo	holdingregdouble	float	32219	-2147483648	2147483647	0	1
Int32Demo	holdingreg32	integer	32215	-2147483648	2147483647	0	1
PB1	coil	boolean	32100				
PB2	coil	boolean	32101				
PB3	coil	boolean	32102				
PB4	holdingreg	integer	32200	0	6	0	1



HMI Details - Minefield

File Edit View History Bookmarks Tools Help

<http://localhost:8080/statuspages/statushmi.html>

Tag Name	Type	Address	Value	Min	Max	Unit	Scale	Offset
PumpSpeedActual	inputreg	32212	32768	32767	0	1		
PumpSpeedCmd	holdingreg	32210	-32768	32767	0	1		
PumpSpeedRaw	holdingreg	32002	-32768	32767	0	1		
SSAni1	coil	32205						
SSPPGrip	coil	32122						
SSPHorz	coil	32121						
SSPPVert	coil	32120						
ShapeDemo	holdingreg	32202	-1	1	0	1		
SparseStrDemo	holdingregstr16	32233,14						
StripChart1	inputreg	32213	-32768	32767	0	1		
StripChart2	inputreg	32214	-32768	32767	0	1		
Tank1Level	inputreg	32210	-32768	32767	0	1		
Tank1Number	inputreg	32210	0	250	0	2.5		
Tank2Level	inputreg	32211	0	100	0	1		
Tank2Number	inputreg	32211	0	250	0	2.5		

Alarm Configuration

Address	Tag Name	Zone List
32400	PB1Alarm	zone1
32401	PB2Alarm	zone2
32402	PB3Alarm	zone3

Event Configuration

Address	Tag Name	Zone List
32300	PumpRunning	zone3
32301	PumpStopped	zone3
32302	Tank1Empty	zone1
32303	Tank1Full	zone1, zone2
32304	Tank2Empty	zone2
32305	Tank2Full	zone2

HMI Configuration Details

3.7.2 HMI Parameters

3.7.2.1 HMI ID

This displays the HMI reserved tag information from the HMI configuration. This includes the following:

Item	Configuration Label	Description
Server ID	[serverid]	This is the HMI server id tag defined by the user.
Client Version	[clientversion]	This is the HMI client version tag defined by the user.

3.7.2.2 Address Tag Configuration

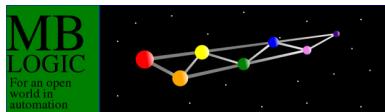


This shows all the normal address tags defined by the user, together with the address, limit, and scaling information. The number of rows will vary according to the number of tags configured.

Item	Configuration Label	Description
Tag Name	N/A	The tag name assigned by the user.
Address Type	addrtype	The address type for the local data table.
Data Type	datatype	The protocol data type.
Memory Address	memaddr	The address in the local data table associated with the tag. ²
Minimum Range	range	The minimum value permitted. ¹
Maximum Range	range	The maximum value permitted. ¹
Scale Offset	scale	The scaling value offset. ¹
Scale Span	scale	The scaling value span. ¹

¹For boolean tags, the range and scale values are not used. The values shown are the defaults, but do not represent actual legal values.

²For string data, the maximum string length is also included and is displayed in brackets. E.g. 2345 (15)



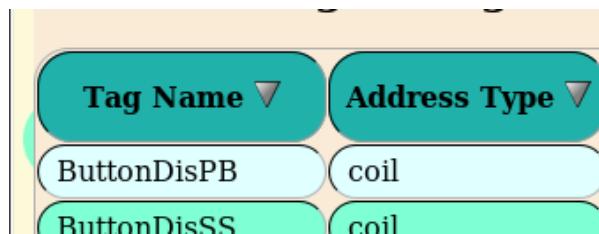
3.7.2.3 Event and Alarm Configuration

Event and Alarm configurations are identical, so both are described here together. These display the event and alarm configurations selected by the user in the HMI configuration. These includes the following:

Item	Description
Address	This is the address in the local data table defined by the user.
Tag Name	This is the event or alarm tag name defined by the user.
Zone List	This is the list of event or alarm zones defined by the user for that event or alarm.

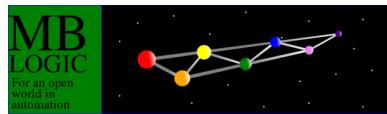
3.7.2.4 Display Order

Each table of data displayed can be sorted by clicking on a table heading. The table will be sorted according to the selected column.



3.7.3 Editing Parameters

3.7.3.1 System Parameters



Server ID:

MBLogic demo server

Client Version:

Ver 0.91 Demo

Change **Cancel**

3.7.3.2 HMI Address Tag Parameters

The number and type of parameters will depend on the the address type.

Tag Name: ButtonDisPB

Memory Address: 32204

Address Type: coil ▾

Change **Cancel** **Delete**

Tag Name: PB4

Memory Address: 32200

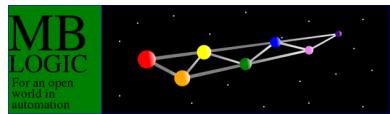
Address Type: holdingreg ▾

Data Type: Boolean Integer Float

Range: 0 Min. 6 Max.

Scale: 0 Offset 1 Span

Change **Cancel** **Delete**



Tag Name:

Memory Address:

Address Type: ▾

String Length:

Change **Cancel** **Delete**

3.7.3.3 Alarms and Events

Zone names must be separated by commas.

Edit Alarms

Memory Address:

Tag Name:

Zone List:

Change **Cancel** **Delete**

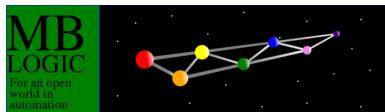
Edit Events

Memory Address:

Tag Name:

Zone List:

Change **Cancel** **Delete**



3.8 Configure Soft Logic IO

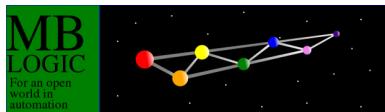
3.8.1 Overview

The "Configure Soft Logic IO" page provides an overview of the current configuration of the soft logic IO subsystem. The soft logic IO sub-system is responsible for transferring data between the soft logic sub-system and the system data table. This is divided into the following sections:

- System parameters
 - Soft logic memory save parameters.
 - Soft logic IO address configuration.
-

The screenshot shows a web browser window titled "Soft Logic IO Details - Minefield". The URL is <http://localhost:8080/statuspages/statusplcio.html>. The page has a header with the MB LOGIC logo and navigation links: Home, Configure, Control, Monitor Data, System, XRef, Help. The main content area is titled "Soft Logic IO Configuration Details". It includes sections for "View Only: Edit:" (radio buttons), "Soft Logic System Parameters" (with fields for Soft Logic Engine Type (ck), Program File (plcprog.txt), and Target Scan Rate (msec) (52)), "Soft Logic Memory Save Parameters" (with fields for Update Interval (sec) (2.1) and Memory Save Addresses (DD1, DF2, DS10)), and "IO Configuration" (a table with columns: IO Section, Address Type, Server Base Address, IO Action, Soft Logic Addresses). The IO Configuration table contains the following data:

IO Section	Address Type	Server Base Address	IO Action	Soft Logic Addresses
Alarms	coil	32400	write	Y30, Y31, Y32, Y33, Y34, Y35, Y36, Y37, Y38
Events	coil	32300	write	Y20, Y21, Y22, Y23, Y24, Y25, Y26, Y27, Y28
HRFloat32Read	holdingregfloat	32217	read	DF10
HRFloat64Read	holdingregdouble	32219	read	DF13
HRInt32Read	holdingreq32	32215	read	DD10



3.8.2 Soft Logic Parameters

3.8.2.1 Soft Logic System Parameters

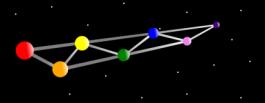
Item	Configuration Label	Description
Soft Logic Engine Type	type	This is the type of soft logic engine enabled for this application.
Program File	plcprog	This is the file containing the soft logic program.
Target Scan Rate (msec)	scan	This is the target (not actual) delay between scans for the soft logic program.

3.8.2.2 Soft Logic Memory Save Parameters

Item	Configuration Label	Description
Update Interval (sec)	updateinterval	This is the minimum interval (in seconds) between saving memory addresses.
Memory Save Addresses	wordaddr	These are the memory addresses which will be saved.

3.8.2.3 IO Configuration

Item	Configuration Label	Description

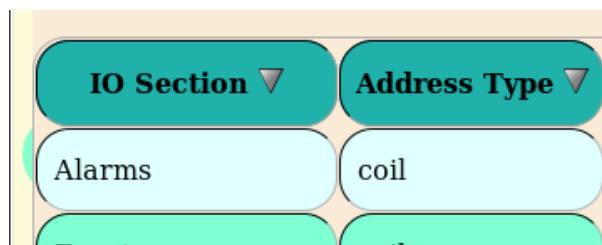


IO Section	[userdefinedname]	This is the IO section with a user defined name.
Address Type	addrtype	This is the system (server) data table address type.
Server Base Address	base	This is the starting address in the system data table used a source or destination for data. ¹
IO Action	action	This determines the direction in which data is copied.
Soft Logic Addresses	logitable	This is the list of individual soft logic addresses used as the source or destination for data.

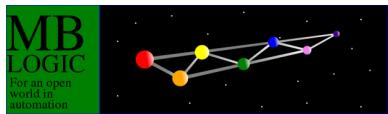
¹For string data, the maximum string length is also included and is displayed in brackets. E.g. 2345 (15)

3.8.2.4 Display Order

The table of data displayed can be sorted by clicking on a table heading. The table will be sorted according to the selected column.



3.8.3 Edit Parameters



3.8.3.1 System Parameters

Scan rate is in milli-seconds.

Soft Logic System Parameters

Program File:

Scan Rate (msec):

Change **Cancel**

3.8.3.2 Memory Parameters

Memory address must be separated by commas.

Soft Logic Memory Save Parameters

Memory Save: Disable Enable

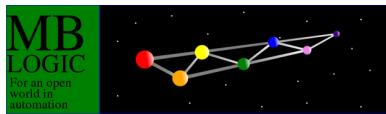
Update Interval (sec):

Memory Addresses:

Change **Cancel**

3.8.3.3 Logic IO Parameters

Soft logic addresses must be separated by commas.



Soft Logic IO Parameters

IO Tag:

Address Type: ▾

Server Base Address:

IO Action: Read Write

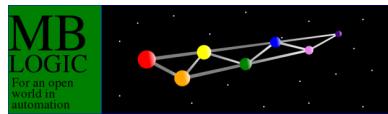
Soft Logic Addresses:

3.9 Soft Logic Program

3.9.1 Overview

The "Soft Logic Program" page provides a summary of the subroutines in the current soft logic program, as well as a means to add, edit, or delete the soft logic program. The first section includes a table with the following items:

- The name of the subroutine
 - A link to a new page which allows the IL to be edited.
 - A check box which allows a subroutine to be selected for deletion.
-



The screenshot shows the MBLogic software interface. At the top, there's a navigation bar with links for Home, Configure, Control, Monitor Data, System, XRef, and Help. Below that is a toolbar with icons for Home, Configure, Control, Monitor Data, System, XRef, and Help. The main content area is titled "Program". It displays a list of subroutines in a grid format:

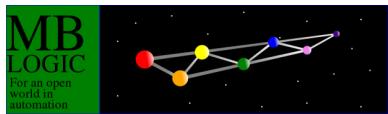
Subroutine	Edit IL	Delete
Alarms	Edit IL	<input type="checkbox"/>
Events	Edit IL	<input type="checkbox"/>
ExtData	Edit IL	<input type="checkbox"/>
LadderDemo	Edit IL	<input type="checkbox"/>
PickAndPlace	Edit IL	<input type="checkbox"/>
StripChart	Edit IL	<input type="checkbox"/>
TankSim	Edit IL	<input type="checkbox"/>
main	Edit IL	<input type="checkbox"/>

Below the list are two buttons: "Delete" and "Refresh Page". At the bottom, there's a section titled "New Subroutine:" with a text input field and a "Add Subroutine" button. A "Done" button is also present.

3.9.2 Editing a Subroutine in IL

To edit a subroutine in IL format, click on one of the links in the "Edit IL" column. To save the subroutine, select the "save" button below the edit window. If the changes were Ok, the subroutine will be inserted into the running program. If they were not Ok, the running program will not be affected and the errors will be listed above the edit window.

Subroutine comments go in the small edit box immediately below the subroutine name.



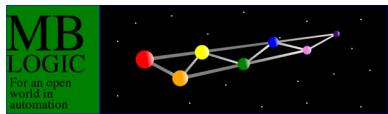
3.9.2.1 Import Points About Editing

Important! It is possible for multiple persons to edit the same program simultaneously provided they are not simultaneously editing the same subroutine. However, the system will not prevent two or more persons from attempting to edit the same subroutine at the same time. If this happens, the last person to save will overwrite previous edits to that subroutine by other users.

Important! The IL editor will automatically add the subroutine declaration when you save the subroutine. Do not add a subroutine declaration manually.

If you were editing a subroutine and have changed its name, the editing system will not be able to find the old name when it attempts to continue editing it. It will therefore display an error. In this case, close the edit page, refresh the list (to update it with the new subroutine name), and select the subroutine again under the new name.

Important! The subroutine edit page only displays errors which are within that subroutine. It does not check calls to other subroutines nor does it check if the rest of



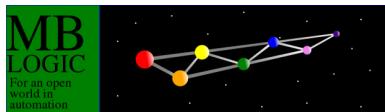
the program calls that subroutine (this may be affected if you rename the subroutine). To check for these conditions, refresh the "Control" page and check for any warnings at the bottom of the page.

3.9.3 Adding and Deleting a Subroutine

To add a subroutine, enter the name of the subroutine in the input box at the bottom of the page and press the "Add Subroutine" button. A new subroutine will be created and added to the list where you can choose to view or edit it.

To delete one or more a subroutines, select the subroutine (tic the corresponding box in delete column) and press the "Delete" button. The subroutine(s) will be deleted and the list updated.

Important! Deleting a subroutine does not delete any calls to the subroutine which may be located elsewhere. Calling a subroutine which does not exist (has been deleted) will cause an error. It is generally advisable to delete all calls to the subroutine first before deleting the subroutine itself.



3.10 Control

3.10.1 Overview

The "Control" page allows changes to the system while it is running.

3.10.2 Configuration Reload

Configurations may be reloaded from disk while the system is running. The following items may be reloaded while running:

- HMI
- Soft logic IO
- Soft logic program

To change the communications configuration, the system must be restarted.

The "Control" page differs from the "Configure" page in that it does not immediately affect the running copy of a configuration. When you reload a configuration (or soft logic program) the system checks it for errors. If any errors were encountered, the old copy continues running (and that is the copy which is shown in the "Configure" page). If no errors were found, the new copy becomes the running copy.

To reload a configuration, select the desired configuration(s) and then press the "Reload File" button. If any errors are found, they will be displayed below. If a very large number of errors are present, only the first few will be shown.

To update the page, press the "Refresh Page" button.

System Control - Minefield

File Edit View History Bookmarks Tools Help

http://localhost:8080/statuspages/statuscontrol.html

System Control

MB LOGIC for an open world in automation

Control

Reload a configuration while the system is running. Any errors will be automatically displayed below.

System	Status	Select
Communications	Ok	
HMI	Ok	
Soft Logic IO	Ok	
Soft Logic	Error	

Reload File

Refresh Page

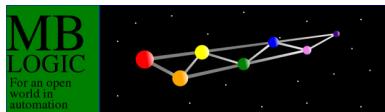
Restart or shutdown the control system.

Restart System Shutdown System

Soft Logic Program Errors:

Error in Events network 6 - missing or incorrect parameters for OUT X25

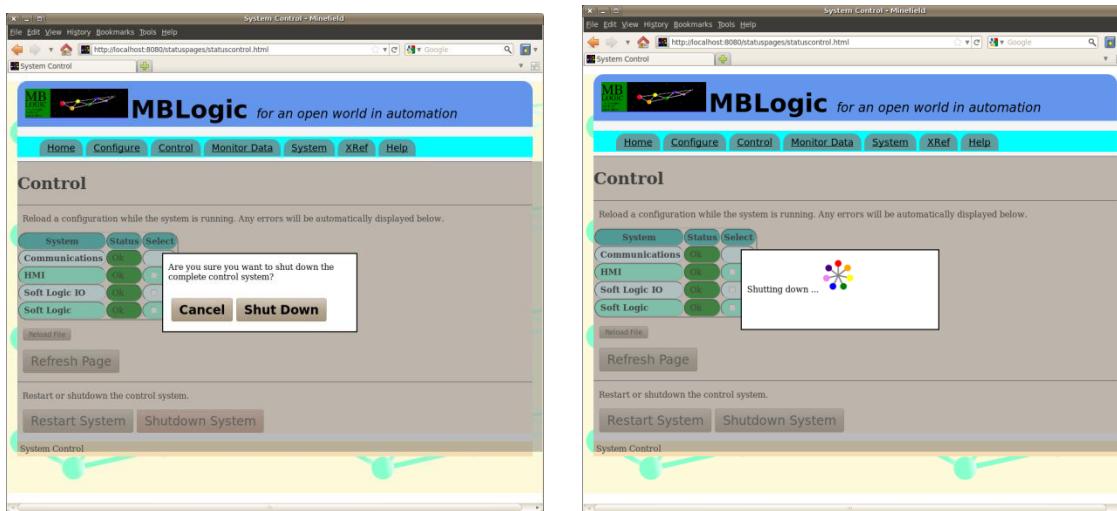
System Control



3.10.3 System Shutdown

The running system may be shut down. To shut down the system, click the "Shutdown System" button. A confirmation dialog will then appear. To cancel the shutdown, press the "Cancel" button. To confirm the shutdown, press the "Shut Down" button. No change will take place until one of these buttons is pressed. The other controls will be disabled.

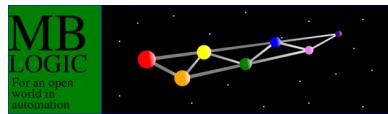
If the shut down request is confirmed, a message will appear indicating the shutdown message has been sent to the system. The message will automatically clear after a time delay.



After the system has shut down, the web interface can no longer function (including loading web pages) as there is no server running for it to communicate to.

3.10.4 System Restart

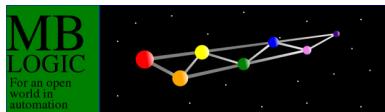
The running system may be restarted. This performs a complete shut down followed by an immediate restart. This may take several seconds. To restart the system, click the "Restart System" button. A confirmation dialog will then appear. To cancel the restart, press the "Cancel" button. To confirm the restart, press the "Restart" button.



No change will take place until one of these buttons is pressed. The other controls will be disabled.

If the restart request is confirmed, a message will appear indicating the restart message has been sent to the system. The message will automatically clear after a time delay.





3.11 Monitor Data and Program

3.11.1 Overview

The "Monitor Data and Program" pages allow monitoring of the following items:

- The contents of selected addresses in the soft logic data table. This is the data table which is directly used by the soft logic program.
 - The contents of selected addresses in the system data table. This is the data table which links the soft logic, HMI, communications, and other systems.
 - The current soft logic program in IL format.
 - The current soft logic program in ladder format. The ladder display can be animated with live data to provide on line monitoring of the soft logic program.
-

The main "Monitor Data" page provides access to additional monitoring pages. Links to pages for viewing the soft logic and system data tables are located in the section titled "Monitor Data Table".

Below that is a section titled "Monitor Soft Logic Program Online". This provides a list of the subroutines in the currently running program, together with links to the ladder and IL monitoring pages.

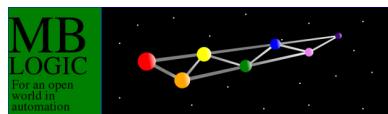
3.11.2 Monitor Data Table

To monitor addresses in the soft logic data table, enter the desired addresses in the data entry boxes (with one address per box). Next, click on the "monitor" button to continuously poll the system for the requested data and display it on the page. Click on "stop" to stop the update, or "single scan" to request a

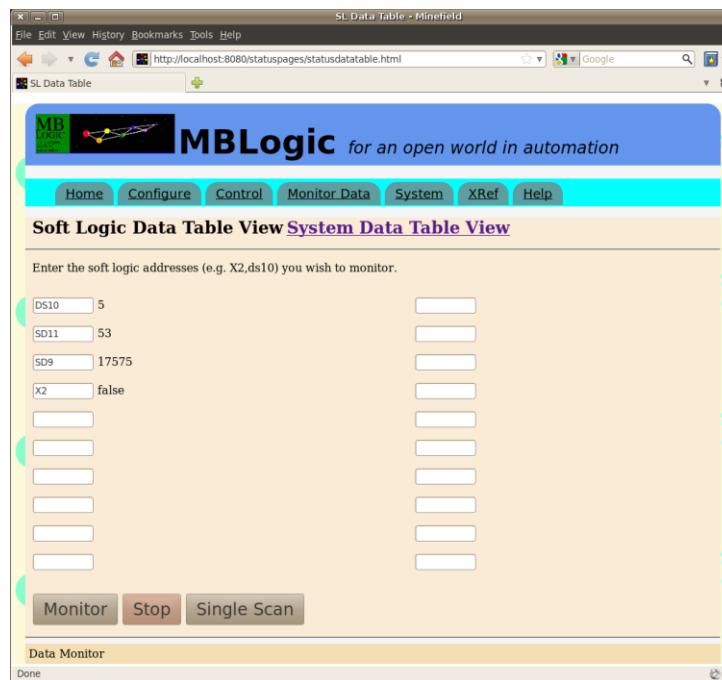
The screenshot shows a web browser window titled "Data Monitor - Minefield". The URL is "http://localhost:8080/statuspages/statusmonitor.html". The interface has a blue header bar with the MB Logic logo and the text "for an open world in automation". Below the header is a navigation menu with links: Home, Configure, Control, Monitor Data, System, XRef, and Help. The main content area is divided into two sections:

- Monitor Data Table**: A table with two columns: Subroutine and Monitor Ladder. The subroutines listed are Alarms, Events, ExtData, LadderDemo, PickAndPlace, StripChart, TankSim, and main. Each row has a "View IL" link next to the "Monitor Ladder" link.
- Monitor Soft Logic Program Online**: A table with three columns: Subroutine, Monitor Ladder, and View IL. The subroutines listed are the same as in the first table. Each row has a "View IL" link next to the "Monitor Ladder" link.

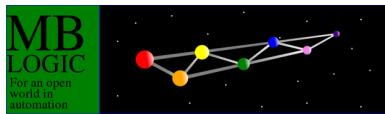
A vertical sidebar on the left contains the text "Page 1 of 1" and "Done".



single update (not continuous) only. If Monitor is selected, updates will continue until either "stop" is selected, or the page is closed.



The system data table can be monitored in a similar manner, with the addition of drop-down menus to select the address type.



Screenshot of the MB Logic System Data Table interface. The window title is "System Data Table - Minefield". The URL in the address bar is "http://localhost:8080/statuspages/statusysdatatable.html". The main content area shows a table with two columns: "Address Type / Address / Value". The left column contains entries for Coil (158, false), Discrete Input (99, false), Holding Register (32210, 0), and Input Register (32210, 5). The right column contains eight empty rows for monitoring. At the bottom are buttons for "Monitor", "Stop", and "Single Scan".

3.11.3 Monitor Ladder

The "monitor soft logic program" section allows viewing the current soft logic program in ladder or IL format, and also monitoring the execution of the soft logic ladder program on line.

3.11.3.1 Viewing a Subroutine in Ladder

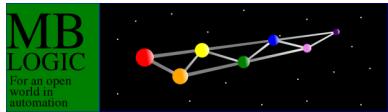
To view a subroutine in IL format, click on one of the links in the "Monitor Ladder" column. This will display the selected subroutine in ladder format. Any rungs which are not in valid ladder format will be displayed in IL.

On line monitoring can be enabled by selecting the "monitor" button.

Screenshot of the Ladder Display interface. The window title is "Ladder Display - Minefield". The URL in the address bar is "http://localhost:8080/statuspages/laddermanager.xhtml?subname=LadderDemo". The main content area shows a ladder logic program with three rungs:

- Rung 1:** A simple rung with contacts C1 and T1 in series, followed by a coil C100 labeled "OUT".
- Rung 2:** A rung with a coil C100 labeled "SET". It includes a note: "Demonstrate different ladder instructions. This subroutine doesn't actually do anything useful." and "A simple rung."
- Rung 3:** A more complex rung with contacts T5, C1, C2, DS100, 50, C100, and Y20, followed by a coil C101 labeled "SET". It includes a note: "More complex rungs are also possible."

At the bottom are buttons for "Monitor", "Stop", and "Single Scan".



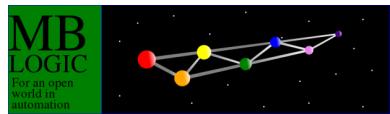
This will cause the page to poll the system for data on a regular basis. Instructions which "true" will change to a different color. Instructions which are "false" will be the default color. Selecting "stop" will stop the on-line monitor. Selecting "single scan" will result in a single update.

Only Boolean instructions (including Boolean compare instructions) can be monitored in ladder format. To see the values present in word instructions, use the data table monitor page.

If a subroutine is changed while the page is displayed, the page will automatically download the new subroutine data and re-display the subroutine using the updated subroutine.

3.11.3.2 Viewing a Subroutine in IL

To view a subroutine in IL format, click on one of the links in the "View Instruction List" column.



IL Display - Minefield

File Edit View History Bookmarks Tools Help
http://localhost:8080/statuspages/ildisplay.html?subname=Alarms Google

IL Display

MBLogic for an open world in automation

Home Configure Control Monitor Data System XRef Help

View IL for: Alarms

Check the alarms.

```
NETWORK 1
// PB1 was pushed.
STR X1
OUT Y30

NETWORK 2
// PB2 was pushed.
STR X2
OUT Y31

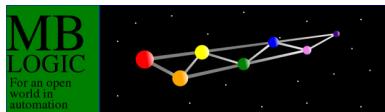
NETWORK 3
// PB3 was pushed.
STR X3
OUT Y32

NETWORK 4
// Return to the main program.
RT
```

Soft Logic Program IL Display

```
graph LR
    N1((Network 1)) --- C1_1(( ))
    N1 --- C1_2(( ))
    N1 --- C1_3(( ))
    N1 --- O1[Output]
    N2((Network 2)) --- C2_1(( ))
    N2 --- C2_2(( ))
    N2 --- O2[Output]
    N1 --- N2
    style N1 fill:#e0f2e0
    style N2 fill:#e0f2e0
```

Done



3.12 System Monitor

3.12.1 Overview

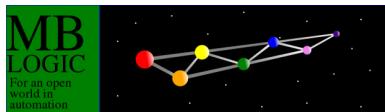
The "System Monitor" page allows monitoring of both the soft logic run status and the status of the communications system. If "Single Scan" is selected, the page will conduct one scan and then update itself. If "Monitor" is selected, the page will poll the server repeatedly and update itself continuously. "Stop" will stop the continuous scan mode.

3.12.2 Program Run Status

The "Program Run Status" displays information about the running soft logic system. This includes the following:

- Program run mode.
- Number of instructions in the program.
- The scan count and scan timers from the soft logic system. The maximum scan will be affected by events such as reloading the soft logic system (which happens between scans).
- The program exit code. This is a code returned from the soft logic system which indicates why the latest program scan ended.
- Exit subroutine. This is the subroutine which the soft logic program was executing when it exited the scan.
- Exit rung. This is the rung (network) which was executing when the soft logic program exited the scan

The screenshot shows the "System Monitor - Minefield" page. At the top, there's a navigation bar with links for Home, Configure, Control, Monitor Data, System, XRef, and Help. Below the navigation bar, the main content area is titled "System Monitor". It features several sections: "Program Run Status" (Program run mode: Running, Number of instructions: 403), "Scan Times" (Scan Count: 6468, Current Scan Repeat Time (msec): 60, Minimum Scan Repeat Time (msec): 53, Maximum Scan Repeat Time (msec): 291), "Program Exit Codes" (Exit Code: normal_end_requested, Exit Subroutine: main, Exit Rung: 16), "Communications" (Server Status: Done, Client Status: Done), and a "Done" button at the bottom.



3.12.3 Communications

The "Communications" status shows the current status of the active servers and clients.

3.12.3.1 Server Status

The "Server Status" table shows all the active servers. The "Connections" and "Rate" columns provide an indication of connection activity. Different protocols will provide different information in these columns depending on the nature of the the protocol.

- modbustcp - Maintains a count of open connections.
- generic - Maintains a count of clients based on a watchdog time-out for each generic client to determine if the client is actively scanning the generic server.
- hmi - Measures connection rate
- status - Measures connection rate
- mbrest - Measures connection rate.
- help - Does not provide any information.

Connection rate is measured as a moving average of requests over a 5 minute period.

3.12.3.2 Client Status

The "Client Status" table shows the current status of the TCP and generic clients. The table includes the connection status, the command status, and a link to a page showing more details. Connection and command status are explained in the section on communications details.

The screenshot displays the "System Monitor - Shirotoke" application window. It includes the following sections:

- Program Run Status:** Shows "Program run mode: Running" and "Number of instructions: 406".
- Scan Times:** Displays scan counts and times for various parameters.
- Program Exit Codes:** Lists exit codes, subroutines, and rungs.
- Communications:** A section for monitoring server and client connections.
- Server Status:** A table showing server names, connections, and rates.
- Client Status:** A table showing client names, connection statuses, command statuses, and log links.

Server Name	Connections	Rate
Generic	0	
HMI	39	
Main1	8	
Web1	0	
WebHelp	0	
status	171	

Connection Name	Connection Status	Command Status	View Log
Beacon	Running	No errors	[View]
Conveyors	Running	No errors	[View]
FanCommand	Running	No errors	[View]
FanSpeed	Running	No errors	[View]
Horn	Running	No errors	[View]
PumpPressure	Running	No errors	[View]
Robot1	Running	No errors	[View]
SlidePosition	Running	No errors	[View]
CasClient	Not Started	No results	[View]

3.12.4 Communications Monitor

The communications monitor shows the status details for an individual communications client. It can also monitor in continuous or single scan mode.

3.12.4.1 *Connection Status Summary*

This shows a summary of the state of the client. The connection status is the state of the overall connection. This may have the following states:

- Starting
- Running
- Stopped
- Faulted

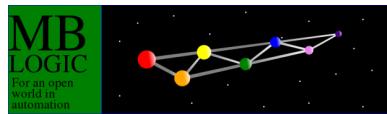
3.12.4.2 *Command Status Summary*

This is a summary of the command status. If there are multiple communications commands, this will show a combined view of them. If all commands are "Ok", the summary will also show "Ok". If one of the commands is not Ok, the summary will show that status. If more than one command is not Ok, the summary will show the status of one of them.

3.12.4.3 *Command Status*

This shows the status of each of the individual commands. Each communications command is shown separately. The command status messages include:

- No errors - No errors were detected.
- Device error - An unspecified error appears to have occurred in the field device.
- Client connection lost - The connection to the client was lost.
- Message time out - Communications timed out.
- Undefined server error - An unspecified error occurred in the communications system.



- No result - There is no valid data to report (the command may not have executed yet).
- Other, protocol specific errors.

3.12.4.4 Connection and Command Events

The connection and command event logs show the connection and command status in a running log format together with a time stamp. Only changes of state are recorded.

3.12.4.5 Client Messages

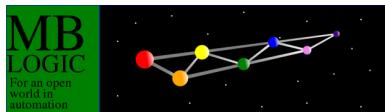
The client messages table shows a running log of messages sent by the client. If there are no client messages, this table will be hidden from view.

The screenshot shows the 'Communications Monitor' window with the following details:

- Header:** MBLogic for an open world in automation
- Toolbar:** Home, Configure, Control, Monitor Data, System, XRef, Help
- Main Area:** Communications Monitor
- Buttons:** Monitor (highlighted), Stop, Single Scan
- Section: Client Status Log Messages for: Beacon**
 - Summary:** Connection Status: Running, Command Summary: No errors
 - Command Status:** Command Name: writecoils, Status: No errors
- Section: Connection Events**

Time	Event
Tue 31 Aug 2010 12:04:47 AM EST	Starting
Tue 31 Aug 2010 12:04:48 AM EST	Connected
- Section: Command Events**

Time	Command	Event
Tue 31 Aug 2010 12:04:48 AM EST	writecoils	No errors



3.13 Cross Reference

3.13.1 Overview

The "Cross Reference" page provides a cross reference of the soft logic program. The address cross reference shows addresses, constants, and subroutine names. The instruction cross reference shows instructions. Each cross reference shows the referenced items as a series of buttons. Press one of the buttons to see more details for that item.

The address cross reference categorizes items by first character. Case is significant, so for example address "DS100" and subroutine name "datasub" are categorized separately. Details for the address cross reference will list the name (e.g. address), subroutine location, and rungs (networks) where they occur.

The instruction cross reference details will show the subroutine and rung (network) where the instruction is used.

The screenshot shows the MB Logic XRef interface. At the top, there's a navigation bar with links for Home, Configure, Control, Monitor Data, System, XRef (which is active and highlighted in blue), and Help. Below the navigation bar is a banner with the MB Logic logo and the tagline "for an open world in automation".

XRef

Address Cross Reference:

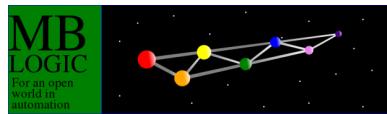
The following is a cross reference of data table addresses, subroutine names, and constants. Items are classified by their first character.

=	0	1	2	3	4	5
6	9	A	C	D	E	L
P	S	T	X	Y	d	m

Instruction Cross Reference:

Cross references showing where instructions are used.

AND	ANDE	ANDGE	ANDGT	ANDLE	ANDLT	ANDN
ANDND	ANDNE	ANDPD	ANDSTR	CALL	CNTD	CNTU
COPY	CPYBLK	END	ENDC	FILL	FINDEQ	FINDGE
FINDGT	FINDIEQ	FINDIGE	FINDIGT	FINDIE	FINDILT	FINDINE
FINDLE	FINDLT	FINDNE	FOR	MATHDEC	MATHHEX	NEXT
OR	ORE	ORGE	ORGTE	ORLE	ORLT	ORN
ORND	ORNE	ORPD	ORSTR	OUT	PACK	PD
RST	RT	RTC	SET	SHFRG	STR	STRE
STRGE	STRGT	STRLE	STRLT	STRN	STRND	STRNE



XRef - Minefield

File Edit View History Bookmarks Tools Help

http://localhost:8080/statuspages/statusxref.html

MBLogic for an open world in automation

Home Configure Control Monitor Data System XRef Help

XRef

Address Cross Reference:

The following is a cross reference of data table addresses, subroutine names, and constants. Items are classified by their first character.

*	0	1	2	3	4	5
6	9	A	C	D	E	L
P	S	T	X	Y	d	m
S						

Address Xref Details

Address	Subroutine	Rung/Network #
C1	LadderDemo	1,2,6,16
C10	LadderDemo	5
C100	LadderDemo	1,2
C101	LadderDemo	2,3
C102	LadderDemo	3
C103	LadderDemo	3

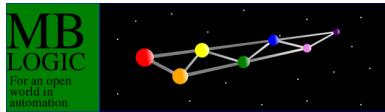
Done

3.13.2 Display Order

The address cross reference table of data displayed can be sorted by clicking on a table heading. The table will be sorted according to the selected column. The address cross reference details table can be sorted by address or subroutine name (but not by rung/network #).

The instruction cross reference details table will be automatically sorted by subroutine name when displayed, but the sort order cannot be changed.

Address Xref Details		
Address	Subroutine	Rung/Network #
X1	Alarms	1
X1	main	1
X2	Alarms	2



4 Communications

4.1 Overview:

The communications subsystem permits access to field devices, HMIs, other control systems, and to MRP/ERP systems. The system provides multiple servers (one for each protocol) and clients (multiple clients per protocol). The number, type, and addresses used for servers and clients is determined by a configuration which is set by the user.

The following protocols are supported in this version:

- Modbus/TCP Server.
 - Modbus/TCP Client.
 - MB-REST - A Modbus-like web service as a server.
 - HMI Web Service server.
 - Generic Client Interface
-

4.1.1 Communications Help Topics:

4.1.1.1 Configuration and Trouble Shooting

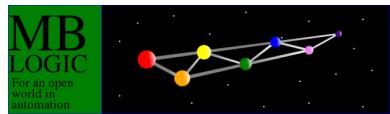
- [Server Communications Configuration](#)
- [Client Communications Configuration](#)
- [Generic Client Protocols](#)
- [Communications Faults](#)
- [Communications Monitoring and Trouble Shooting](#)

Also see the "System Status" documentation for information on how to use the web interface to view and edit configurations.

4.1.1.2 Expanded Register Map

- [Expanded Register Map](#)

4.1.1.3 Protocols



- [Modbus/TCP Server and Client Protocol](#)
- [MB-REST Web Service Protocol](#)
- [HMI Protocol](#)
- [ERP Protocol](#)

4.1.1.4 Generic Clients

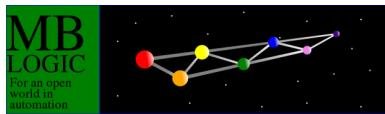
- [Generic Clients](#)
- [Generic Client Framework](#)
- [Generic Client User Protocol Class](#)

4.1.1.5 Services

- [RSS Monitor Feed](#)

4.1.1.6 Communications Basics

- [Modbus/TCP Basics](#)



4.2 Server Communications Configuration

4.2.1 Overview

Servers configurations are saved in a text file called "*mbserver.config*". This configuration file is read on start up of the software and used to determine what protocols to enable, what network addresses and ports to use, what data to read or write, etc. This file must be edited in order to configure a system for every individual application.

4.2.2 Configuration Overview

4.2.2.1 *Editing a Configuration*

There is one, and only one server communications configuration file for each system. This file is in ASCII text format and may be edited with any standard text editor. If you use a word processor, be sure to save in a plain text mode which does not insert formatting characters into the text.

The status system also provided a web based forms for editing the configuration rather than using a text editor.

4.2.2.2 *Configuration Elements*

A communications configuration file contains the following elements:

- Server name - This allows a name to be assigned to the system and displayed on status web pages.
- Server configurations - This configures the server names, protocol type, and IP port numbers.
- Comments - Comments may be included in the configuration file to assist in documenting the particular application configuration.

4.2.2.3 *Configuration Syntax*

The configuration syntax has the following general properties:



- Sections. Sections begin a series of related configuration values. A section name is an identifier which is enclosed by square brackets ("[]"). A section includes all items until the beginning of a new section, or until the end of the file. Section names must be unique and must not be duplicated in any other section. Duplicate section names are ignored.
- Configuration items. Any line following the beginning of a section is considered to be an item which belongs to that section (excluding comments). Configuration items are key/value pairs in the format "key=value".
- Each item must be on a separate line.
- Keys are separated from values by equal signs ("="). Because equal signs are used as key/value separators, they may not be used as part of any key or value.
- Parameters are case sensitive. All keys and values must be in the correct case. For example, "Main1", "main1", and "mAin1" are three different identifiers.
- All keys or values which are defined by the configuration syntax are always in lower case.
- User defined values may be in upper, lower, or mixed case.
- The ampersand ("&") character has special meaning. Certain sections are predefined by the configuration system. The names for these will start with an ampersand. User defined section names may not start with an ampersand. Command item names must start with an ampersand. Any item starting with an ampersand is assumed to be a command.

4.2.2.4 Comments

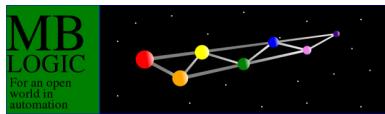
Any line beginning with a '#' character is considered to be a comment and is ignored. Comments must begin at the start of a line, and may not be on the same line as any other valid input.

4.2.2.4.1 Example

```
# 16-Apr-2008. This is a comment.  
# This is also a comment.
```

4.2.3 System Configuration

The system configuration contains parameters which are system wide as opposed to limited to a specific protocol.



4.2.3.1 Server ID Name

The server id name is the name applied to the complete system (not just an individual protocol server). The server id name may be any arbitrary text, including spaces. The server id name appears in a special section which is always called "&system" (note the ampersand). The actual name used for the server is then indicated by a key called "serverid". The server id name is in the following format:

4.2.3.2 Expanded Holding Register Map Offsets

The expanded register map configuration allows certain server protocols to access additional memory beyond the normal protocol limits. Details of this may be found in the separate section on expanded register maps. This section deals with the configuration parameters.

By default, expanded register map support for servers is turned off. If you do not configure the parameters, unit IDs will simply be ignored by the servers. If you configure the parameters incorrectly, the expanded register map feature will also be disabled, as the system will not be able to calculate the address offsets correctly.

There are two parameters relating to expanded register maps. These are:

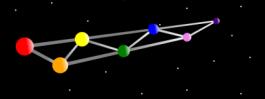
- **mbaddrxp** - This enables the feature.
- **mbuidoffset** - This sets the numeric UID offset and factor.

The **mbaddrxp** parameter must be set to "incremental" to enable the expanded register map. Any other value will act to disable it. In future other algorithms may be introduced, but at present only "incremental" is available.

The **mbuidoffset** has two values which must be separated by a comma. These are (in order) the "UID offset" and "UID factor". Details of how to set these factors are discussed in the section on expanded register maps.

Both values must be positive integers. The UID offset must be less than 255. The UID factor must be less than the maximum holding register memory map address.

4.2.3.3 Example



```
[&system]
serverid=Name of Example Server
mbaddrrep=incremental
mbuidoffset=1,65536
```

4.2.4 Server Configuration

4.2.4.1 Configuration Items

Server configurations consist of three elements:

Item	Description
The section name.	This is used as the name of the server. This name is used to identify the server in the status monitoring system and will appear on the status web pages.
type	This is the protocol type and is used to determine the communications method used. This must be one of the recognised protocol types (listed below).
protocol	This determines the protocol recognised by the server. This must be one of the recognised protocols (listed below).
port	The IP port number. This must be a valid integer IP port number.

4.2.4.2 Example

```
[Main1]
type=tcpserver
protocol=modbustcp
port=8502
```

4.2.4.3 Types

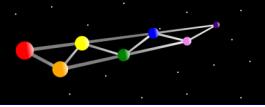
The recognised types are:

- `tcpserver`

4.2.4.4 Protocols

The recognised protocols are:

Protocol	Description
<code>modbustcp</code>	The Modbus/TCP protocol.
<code>mbrest</code>	A Modbus-like web service protocol (obsolete).
<code>mbhmi</code>	The server used to serve HMI web pages, and also provide a web service protocol used by the HMI system.
<code>rhmi</code>	A restricted version of the "mbhmi" protocol. It uses the same configuration and resources as the "hmi" option, but write and alarm acknowledge commands are disabled.
<code>erp</code>	Another restricted version of the "mbhmi" protocol, intended for MRP/ERP and other business integration applications. It uses the same configuration and resources as the "hmi" option, but only supports "read" and "write" commands from a specified (filtered) tag list.
<code>status</code>	The status web server. This is not a communications server in the same sense that the others are, but this allows the IP port number to be configured. If this server is not configured, the status web pages will not be available.
<code>help</code>	This is similar to the status protocol, in that it sets the address used by a web server. In this case, this is the web server used for the "help" system.



generic

This is a special server protocol used to support generic clients. You must enable this if you wish to make use of a generic client.

Only one server of each protocol can be configured. There is no limit however to how many incoming client connections each server will accept. If a protocol is not configured, that protocol server will not be active.

4.2.5 Complete Example

The following is an example showing a complete configuration file.

4.2.5.1 Example

```
# Test configuration for MB Server.
# 23-Jul-2010

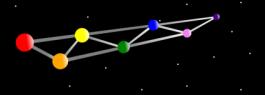
# Name of server.
[&system]
serverid=Loopback Server
# The following parameters are optional.
mbaddrexp=incremental
mbuidoffset=1,65536

# Server configurations.
# Modbus/TCP protocol.
[Main1]
type=tcpserver
protocol=modbustcp
port=8502

# MB-REST web service protocol.
[Web1]
type=tcpserver
protocol=mbrest
port=8085

# MB-HMI web server and web service protocol.
[HMI]
type=tcpserver
protocol=mbhmi
port=8082

# Restricted (read-only) version of the mbhmi protocol.
[ROHMI]
protocol = rhmi
```



```
type = tcpserver
port = 8083

# Limited version of the mbhmi protocol for MRP/ERP applications.
[ERPAccess]
protocol = erp
type = tcpserver
port = 8084

# Status web server.
[status]
type=tcpserver
protocol=status
port=8080

# Help system web server.
[WebHelp]
type=tcpserver
protocol=help
port=8081
```

4.2.6 Loop Back Sample

The sample configuration file supplied with the system uses what is referred to as a "loop-back" configuration for the clients. That is, a set of Modbus/TCP clients are configured to poll the system's own Modbus/TCP server. This is not in itself a useful feature. It is provided mainly to demonstrate the communications features. A different configuration should be created for a practical application. The sample configuration demonstrates the following features.

- The server is named "Loopback Server"
- A Modbus/TCP server is started on port 8502.
- A Modbus-like web service is start on port 8085.
- An HMI server is started on port 8082.
- A status web server is started on port 8080.
- A help system web server is started on port 8081.



4.3 Client Communications Configuration

4.3.1 Overview

Clients configurations are saved in a text file called "*mbclient.config*". This configuration file is read on start up of the software and used to determine what protocols to enable, what network addresses and ports to use, what data to read or write, etc. This file must be edited in order to configure a system for every individual application.

4.3.2 Configuration Overview

4.3.2.1 *Editing a Configuration*

There is one, and only one client communications configuration file for each system. This file is in ASCII text format and may be edited with any standard text editor. If you use a word processor, be sure to save in a plain text mode which does not insert formatting characters into the text.

4.3.2.2 *Configuration Elements*

A client communications configuration file contains the following elements:

- Client connections - This configures the client names, protocol type, address, polling rate, and other information.
- Client commands - This configures the client communications commands, including memory addresses, protocol functions, and other information.
- Fault Configuration - This configures the client communications fault indication, including the memory addresses where the fault information is stored.
- Comments - Comments may be included in the configuration file to assist in documenting the particular application configuration.

4.3.2.3 *Configuration Syntax*

The configuration syntax has the following general properties:



- Sections. Sections begin a series of related configuration values. A section name is an identifier which is enclosed by square brackets ("[]"). A section includes all items until the beginning of a new section, or until the end of the file. Section names must be unique and must not be duplicated in any other section. Duplicate section names are ignored.
- Configuration items. Any line following the beginning of a section is considered to be an item which belongs to that section (excluding comments). Configuration items are key/value pairs in the format "key=value".
- Each item must be on a separate line.
- Keys are separated from values by equal signs ("="). Because equal signs are used as key/value separators, they may not be used as part of any key or value.
- Parameters are case sensitive. All keys and values must be in the correct case. For example, "Main1", "main1", and "mAin1" are three different identifiers.
- All keys or values which are defined by the configuration syntax are always in lower case.
- User defined values may be in upper, lower, or mixed case.
- The ampersand ("&") character has special meaning. Certain sections are predefined by the configuration system. The names for these will start with an ampersand. User defined section names may not start with an ampersand. Command item names must start with an ampersand. Any item starting with an ampersand is assumed to be a command.

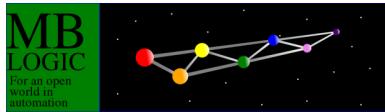
4.3.2.4 Comments

Any line beginning with a '#' character is considered to be a comment and is ignored. Comments must begin at the start of a line, and may not be on the same line as any other valid input.

4.3.2.4.1 Example

```
# 16-Apr-2008. This is a comment.  
# This is also a comment.
```

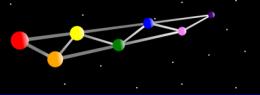
4.3.3 TCP Client Connections



TCP Client connections define the connection parameters to an external server (e.g. an I/O module, valve bank, or RTU). Each client connection executes independently of (is not synchronised with) any other client connect.

4.3.3.1 Configuration Items

Item	Description
The section name.	This is used as the name of the client. This name is used to identify the client in the status monitoring system and will appear on the status web pages.
type	This is the protocol type and is used to determine the communications method used. This must be one of the recognised protocol types (listed below).
protocol	This determines the protocol recognised by the client. This must be one of the recognised protocols (listed below).
host	The IP address of the server the client is polling. If the server is on the same PC, 'localhost' is also a valid address.
port	The IP port being polled.
action	Must always be "poll" at this time.
cmdtime	The time delay in milliseconds between execution of consecutive commands.
repeattime	The time delay in milliseconds between executing the last

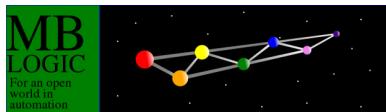


	command and beginning the set of commands again.
retrytime	The maximum time in milliseconds to wait for a reply to a command. If no reply is received, the next command is attempted.
fault_inp, fault_coil, fault_inpreg, fault_holdingreg	The data table addresses to use for indicating faults. These must be valid data table addresses. Faults are discussed in more detail below.
fault_reset	The data table address to use for resetting faults. This must be a valid coil address. Faults are discussed in more detail below.
commands	These are instructions which the client uses to poll the remote server addresses. Commands are discussed in more detail below. There may be any number of client commands.

4.3.3.2 Example

```
[Module002]
type=tcpclient
protocol=modbustcp
host=localhost
port=8502
action=poll
cmdtime=050
repeattime=050
retrytime=5000
fault_inp=10002
fault_coil=10002
fault_inpreg=10002
fault_holdingreg=10002
fault_reset=65281
&command1= function=2, remoteaddr=1, qty=1, memaddr=1002, uid=2
&command2= function=3, remoteaddr=1, qty=4, memaddr=5000
```

The items may appear in any order.



4.3.3.3 Types

The recognised types are:

- tcpserver

4.3.3.4 Protocols

The recognised protocols are:

- modbustcp - The Modbus/TCP protocol.

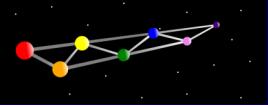
4.3.3.5 Commands

Client commands are the actions to be executed for each client connection.

Commands within a connection are executed consecutively. There is however no guaranty as to the order in which they are executed, and they are not synchronised with commands for other clients.

Client commands are part of the section they are associated with and must be defined as part of that section. The recognised commands parameters are:

Command Element	Description
The command name.	This is used as the key to the item. This name is used to identify the command in the status monitoring system and will appear on the status web pages. The command name must start with an ampersand. No two commands may have the same name in the same client connection (section). If they do, all but one will be ignored. However, it is permitted to use the same name in other client sections, provided that it appears no more than once in the same section. The names may be any combination of upper and lower case and may be descriptive (e.g. "Valves").



function	The client protocol code or name. For Modbus, this is the Modbus function code.
remoteaddr	The memory address at the server being polled.
qty	The number of consecutive addresses being polled. For protocol functions which do not require a quantity, this parameter must be present and a valid number, although the value will not necessarily be used.
memaddr	The memory address in the local data table which is being read or written to.
uid	The unit ID to be sent with the message. This is optional, and if this parameter is left out a default unit ID of 1 is used. Each command may specify a different unit ID independently of any other command. This parameter is optional, but good practice would be to include it even if the value does not differ from the default.

4.3.3.6 Example

```
&command1= function=2, remoteaddr=1, qty=1, memaddr=1002
&command1= function=2, remoteaddr=1, qty=1, memaddr=97345, uid=2
```

The command name must be the first element in the line. Other parameters may appear in any order. All parameters associated with a single command must appear on the same line.

4.3.4 Generic Clients

Generic clients permit adding new clients to the system without making them a permanent part of the software. This is accomplished by running the generic client as

a separate process (program) and having it communicate with the main system via a special protocol. Generic clients are primarily intended to make integrating serial protocols (including custom protocols) into the main system easier, but they are not restricted to serial communications, nor indeed are they restricted to communications at all.

Generic client support is still experimental and subject to change in future.

4.3.4.1 Configuration Items

Item	Description
The section name	This is used as the name of the client. This name is used to identify the client in the status monitoring system and will appear on the status web pages. This also acts to identify the generic client to the server when the client contacts it.
type	Must be "genericclient"
protocol	This determines the protocol information is displayed on the status interface. This may be any value.
clientfile	This is the file name for the generic client. It may also include additional command line parameters which are to be passed to the client on start.
restartonfail	This defines the restart characteristics to be used if the client fails (dies or crashes) during operation.
action	Must always be "poll" at this time.
readtable	This is the list of addresses to be read from the server and passed to the generic client. Details are described below.



writetable	This is the list of addresses to be read from the client and passed to the server. Details are described below.
fault_inp, fault_coil, fault_inpreg, fault_holdingreg	The data table addresses to use for indicating faults. These must be valid data table addresses. Faults are discussed in more detail below.
fault_reset	The data table address to use for resetting faults. This must be a valid coil address. Faults are discussed in more detail below.
commands	These are instructions which the client uses to poll the remote server addresses. Commands are discussed in more detail below. There may be any number of client commands.

4.3.4.2 Client File Name

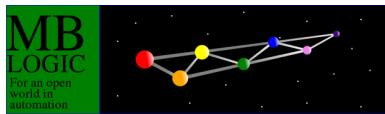
The file name of the generic client is specified by the "clientfile" parameter. This **must** be a valid Python program file. The system will explicitly call the Python interpreter to execute the file.

The file name parameter can include additional command line parameters. These parameters will be passed to the client at start up.

However, the system has parameters of its own which it passes to the client, and any user defined parameters must not interfere with these. These system parameters are:

- c - Client name. This is the name of the client (as defined in the section name).
- p - The generic interface server port number.

4.3.4.3 Restart Characteristics



If a generic client unexpectedly exits ("dies") while running, it can be automatically restarted by setting the appropriate option for "restartonfail". The restart options are as follows:

- yes - Restart the client if it terminates unexpectedly.
- no - **Do not** restart the client if it terminates unexpectedly.
- nostart - Do not start the client at all.

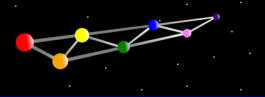
The system monitors configured clients to see if they are running. This check takes place at approximately 1 second intervals. If "restartonfail" is set to "yes", and if the client terminates unexpectedly, the system will attempt to restart it immediately at the next check interval. The system will attempt to restart the client up to 5 times. If the maximum number of restart attempts is exceeded, the system will cease trying to restart that client. If the client was successfully restarted, the restart counter is reset. This means a client can be automatically restarted as many times as necessary, provided the number of **consecutive** unsuccessful attempts is not exceeded.

4.3.4.4 Data Table Read and Write Addresses

A generic client communicates with the main server by copying a selected part of the data table to and from the server. The "readtable" item specifies the addresses to be read from the server and copied to the generic client. The "writetable" item specifies the addresses to be read from the generic client and copied to the main server. They are specified in the format: "<type>=<address>:<length>". The address types may be the following:

- coil - Coil addresses.
- inp - Discrete input addresses.
- holdingreg - Holding register addresses.
- inpreg - Input register addresses.

The addresses may be any valid address for each type. The lengths may be any length which does not result in attempting to read or write beyond the end of the data table. These parameters are not limited by Modbus protocol read or write specifications. Any type may be read from or written to. Any quantity may be transferred at once. However, excessively large data transfers may cause problems with system performance.



In addition to the above types, the type of "none" is also valid. This may be used as a placeholder to specify that no data transfer is requested for that item. Some examples are:

```
# Do not read any addresses
readtable= none
# Write 10 coils starting at 9100, and 4 holding registers starting at
9650
writetable= coil=9100:10, holdingreg=9650:4

# Read and write all types.
readtable= coil=9100:10, inp=9100:11, holdingreg=9650:4,
inpreg=10000:1
writetable= coil=900:10, inp=900:1, holdingreg=62050:150, inpreg=2:99
```

4.3.4.5 Client Configuration Items

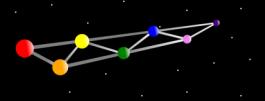
Any parameters which are not recognised by the server as standard generic client items, and which are not commands are passed to the generic client as is. This allows a generic client to define their own parameters as required for the application.

4.3.4.6 Commands

Client commands are the actions to be executed for each client connection. Commands within a connection are executed consecutively. Commands for generic clients are defined by the application and need not be present. The server however will parse them from the parameter set for convenience. The generic client is responsible for validating the command parameters.

4.3.4.7 Example

```
# This is used to help demonstrate the generic client.
[CasClient]
type=genericclient
protocol=cascadas
clientfile=casclient.py -d 1
restartonfail=yes
cmdtime=750
repeattime=1250
retrytime=5000
action=poll
readtable= none
writetable= coil=9100:10, holdingreg=9650:4
fault_inp=10020
```



```

fault_coil=10020
fault_inpreg=10020
fault_holdingreg=10020
fault_reset=65293
&cmd1= PL1, PL2, PL3, PL4, Tank1Level

```

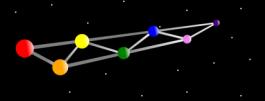
4.3.5 Fault Configuration

Fault configurations are the memory addresses where communications faults which are detected by each client are indicated. There must be only one fault configuration per client.

4.3.5.1 Configuration Items

Item	Description
fault_inp	When a fault is detected, this discrete input will be turned on.
fault_coil	When a fault is detected, this coil will be turned on.
fault_inpreg	When a fault is detected, this input register will be set to a non-zero value. The value will be protocol dependent and may indicate the type of fault.
fault_holdingreg	When a fault is detected, this input register will be set to a non-zero value. The value will be protocol dependent and may indicate the type of fault.
fault_reset	When this coil address is turned on, the fault indication bits and registers are reset to "0", and this coil is turned off. Only the highest (last) 256 coils may be used as fault reset coils. If a lower address is selected no error will be indicated, but the coil will not be monitored to reset the fault indicating registers or bits.

4.3.5.2 Example



```
fault_inp=256
fault_coil=97
fault_inpreg=10000
fault_holdingreg=52000
fault_reset=65281
```

4.3.5.3 Additional Notes

The items may appear in any order.

Fault configurations are part of the section they are associated with and must be defined as part of that section. All of the fault configuration items must appear with each client.

The fault reset coils are scanned once every several seconds. It therefore may take several (e.g. 2 or 3) seconds for the faults to reset once the reset coil is activated.

All four indicating parameters (inp, coil, inpreg, holdingreg) are equivalent in that all are turned on together in the event of any fault. This means that any one of these may be read to determine the presence of a fault. It is suggested that addresses be chosen to allow them to be read as part of the same read operation as other normal data.

4.3.6 Complete Example

The following is an example showing a complete configuration file.

4.3.6.1 Example

```
# Client connections.
# This "loops back" into the server. You don't really want to do this.
[Module000]
type=tcpclient
protocol=modbustcp
host=localhost
port=8502
action=poll
cmdtime=050
repeattime=75
retrytime=5000
fault_inp=15
fault_coil=15
fault_inpreg=10096
```



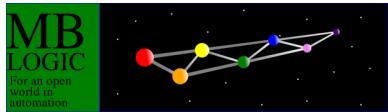
```

fault_holdingreg=10096
fault_reset=65280
&command1=function=1, remoteaddr=0, qty=15, memaddr=0

# This is another client connection.
[Module001]
type=tcpclient
protocol=modbustcp
host=192.168.5.100
port=502
action=poll
cmdtime=040
repeattime=59
retrytime=5000
fault_inp=32
fault_coil=32
fault_inpreg=10097
fault_holdingreg=10097
fault reset=65281
&Sensor1=function=2, remoteaddr=0, qty=16, memaddr=16
&PushButtons=function=1, remoteaddr=0, qty=16, memaddr=16
&PressForce=function=3, remoteaddr=0, qty=2, memaddr=5000
&PumpStatus=function=4, remoteaddr=0, qty=4, memaddr=200
&ValueBank=function=4, remoteaddr=8192, qty=120, memaddr=32768
&DoorSwitches=function=15, remoteaddr=8192, qty=48, memaddr=61000
&Beacon=function=6, remoteaddr=39, qty=1, memaddr=62000

# This is an example of a generic client.
[HartClient]
protocol = hart
type = genericclient
serialport = /dev/ttyS1
clientfile = hartclient.py -d 1
restartonfail = yes
action = poll
cmdtime = 500
repeattime = 1000
retrytime = 1000
fault_coil = 12500
fault_inp = 12500
fault_holdingreg = 12500
fault_inpreg = 12500
fault_reset = 65297
readtable = coil=0:0, holdingreg=0:0, inp=0:0, inpreg=0:0
writetable = coil=11500:20, holdingreg=11500:25, inp=11500:125,
inpreg=11500:100
retries = 0
statisticstable = 60
&readprimaryvar = action=poll, uid=0, function=1, datatype=inpreg,
dataoffset=0
&readuniqueid = action=poll, uid=0, function=0, datatype=holdingreg,
dataoffset=10
&readallvar = action=oneshot, uid=0, function=3, datatype=inpreg,
dataoffset=10

```

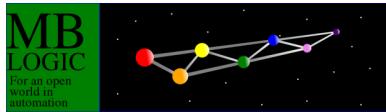


```
&readcurrentrange = action=poll, uid=0, function=2,  
datatype=holdingreg, dataoffset=0
```

4.3.7 Loop Back Sample

The sample configuration file supplied with the system uses what is referred to as a "loop-back" configuration for the clients. That is, a set of Modbus/TCP clients are configured to poll the system's own Modbus/TCP server. This is not in itself a useful feature. It is provided mainly to demonstrate the communications features. A different configuration should be created for a practical application. The sample configuration demonstrates the following features.

- Several Modbus/TCP clients are started, each demonstrating a different configuration. These clients read and write the data table through the Modbus/TCP server (noted above).



4.4 Generic Client Protocols

4.4.1 Overview

The "generic client" mechanism provides a means of adding protocols or other system extensions. Generic clients have direct access to the system data table via the server for generic clients.

Generic client support is experimental at this time and is subject to change.

4.4.2 Supported Protocols

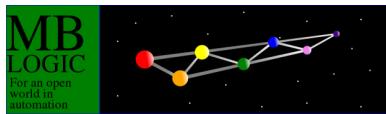
The following generic clients are provided with the system.

- [Modbus/RTU \(serial port\)](#)
 - [Hart \(serial port\)](#)
 - [SAIA Ether-SBus \(Ethernet\)](#)
-

4.4.3 Generic Client Configuration

Generic client configurations include both standard and protocol specific parameters. Configuration parameters are stored in the "mbclients.config" configuration file.

There is more information on configuration as part of "Client Communications Configuration". In addition, there is a configuration form as part of the status configuration system.



4.5 Communications Errors

4.5.1 Overview

Communications errors come in several different types:

- Connection errors.
- Command errors.

Communications errors can be viewed using the "status" web based monitoring interface. They are also reported in the system data table where they may be accessed by the soft logic program or other systems.

4.5.2 Types of Errors

4.5.2.1 *Connection Errors*

Client connection faults are reported in the status monitoring web page and in the data table (more details below). Servers do not initiate connections, so they do not report the inability to create a connection.

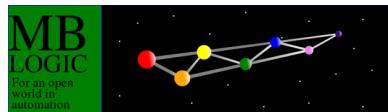
Connection faults codes are protocol dependent, to the degree that they must not duplicate the error codes used for the protocol on that connection.

4.5.2.2 *Command Errors*

Command errors are errors or faults which are defined by the particular communications protocol being used. These are typically caused by either an incorrect configuration or a problem in a remote client. Connection faults are related to problems establishing or maintaining a connection between a local client and a remote server.

4.5.3 Status and Error Codes

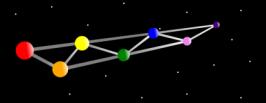
4.5.3.1 *Connection Status and Error Codes*



Code	Description
starting	The connection is in the process of starting.
running	The connection is running normally.
stopped	The connection is stopped.
faulted	The connection is faulted.

4.5.3.2 Client Command Status Codes

Code	Data Table Value	Description
ok	0	No errors
badfunc	1	Unsupported function
badaddr	2	Invalid address
badqty	3	Invalid quantity
deviceerr	4	Device error
frameerr	5	Frame error
connection	255	Client connection lost
timeout	254	Message time out
servererr	253	Undefined server error



badtid	252	TID Error
noresult	251	No result

4.5.4 Error Reporting

4.5.4.1 Web Based Status Reporting

Live fault information is displayed on the status web pages. This includes connection and command errors, as well as a log of past errors. See the documentation on the status web pages for more information.

4.5.4.2 Data Table Reporting

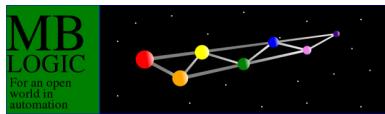
In the communications configuration, the user can specify a set of addresses for each client to be used for reporting client communications errors. Each time an error is detected, these addresses are updated to indicate the error. This error information may be read using normal protocol commands to detect if an error is present, and what the error is.

The same error is reported in all four memory types (coil, discrete input, holding register, input register) in the system data table. This allows an error to be monitored in a manner which is convenient to any configuration.

When an error is detected, the specified coil and discrete input are turned on, and the appropriate numeric code is written to the holding and input registers.

If error information is present, it will not be reset or erased unless specifically requested. However, newer errors will automatically overwrite older error information.

The error reporting addresses may be located anywhere in memory. It may be convenient to place them adjacent to other data which is being read to permit the data and error indication to be read in a single read operation.



Error reporting is on a per-connection basis. This means that if a connection uses multiple commands, then errors associated with any of these commands will be stored in the same set of fault addresses. However, a separate set of addresses should be used for each connection.

4.5.5 Resetting Error Codes

Error indication can be reset in two ways:

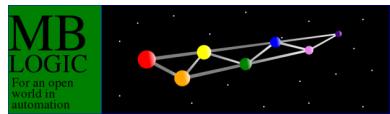
- By overwriting the data table address directly.
- By writing to a fault reset coil.

Addresses used to report faults are normal data table addresses and may be overwritten using normal protocol commands. However, this requires several write operations, and the server protocol may not permit writing to all address types.

In addition to simply overwriting a fault address, a special set of coil addresses has been reserved for resetting fault information. In the data table, a range of coil addresses is regularly monitored to see if any of them have been turned on. If one or more has been turned on, the system will look to see if they have been associated with a particular client fault configuration. If so, the associated fault indication addresses (as well as the reset coil itself) will be reset.

Any coils which are in the monitored range will be turned off regardless of whether or not they are associated with a fault configuration. This means that coils which are not used for fault reset may still not be used for any other purpose. Writing to a coil in the monitored range but which is not configured will not result in an error, but it will result in that coil being automatically reset.

The monitored coils are located in the upper 256 coil addresses (65279 to 65535). They are monitored approximately once per second. This means that it may take up to approximately 1 second for a fault reset to complete once it has been requested.



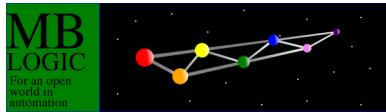
Resetting a fault does not automatically re-initialise any communications. It simply clears the fault reporting memory. It also does not affect the display of faults on the status web pages.

4.5.6 Configuration

Faults reporting is configured via the system configuration file. The following items can be configured:

- The addresses (coil, discrete input, holding register, input register) used for reporting faults.
- The address used for resetting a fault.

See the documentation on system configuration for more details.

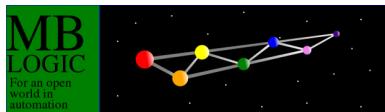


4.6 Communications Monitoring and Trouble Shooting

4.6.1 Overview

The application includes a web based "status monitoring" system. The status system operates on its own port, which may be configured by the user. The status system provides the following features which help when configuring and trouble shooting communications:

- A summary listing of the current configuration, including active servers and clients.
- A summary of the current status of all the clients.
- A web page for each client, showing the configuration and a running log of any errors.
- A listing of any communications configuration errors.
- Web based on-line documentation for the system (including this page).



4.7 Expanded Register Map

4.7.1 Overview

The holding registers have an expanded memory map which extends beyond the limits of the Modbus protocol. Most sub-systems can access these expanded addresses directly because they are not limited by Modbus protocol limitations. However, some protocols have limits which prevent larger addresses from being directly used while still maintaining compatibility with standards. The expanded register map contains a method of allowing some of these protocols to access the expanded map while maintaining compatibility with standards.

4.7.2 Expanded Map Sizes

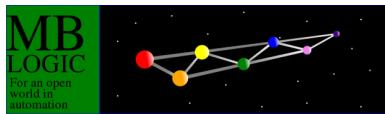
The system data table uses Modbus conventions to allow for simpler configuration of communications. However, the system data table is *not* inherently limited by the Modbus protocol itself. The address limits are:

- Discrete inputs - 0 to 65,535
- Coils - 0 to 65,535
- Input Registers - 0 to 65,535
- Holding Registers - 0 to 1,048,575

The holding register map is larger due to user demand for an increased data table location for storing large amounts of register data. There is no equivalent demand for the other data areas, and the smaller sizes used there avoid using large amounts of RAM. However, even 65,536 data locations is far more than most applications would use.

It is recommended that you do *not* use expanded register map addresses unless you have a clear reason for doing so. The normal address space of 0 to 65,535 is more than adequate (by a large margin) in almost all applications.

4.7.3 Disabling the Expanded Register Map



The server support for expanded register maps is off by default. That is, unless you explicitly configure the expanded map factors, the server will not use the expanded maps and will ignore UIDs.

However, the expanded map address space is always present, and can be accessed directly by those systems that do not have address limitations. This includes the built-in Modbus/TCP clients. Since clients do not use the Modbus protocol to access internal memory they are not limited to 16 bit addresses when accessing the internal data table.

4.7.4 Supported Protocols

At present, only the Modbus/TCP server supports the expanded register map system. The MBRest web service protocol does not support it. The other sub-systems such as the HMI, soft logic, and system monitoring and control do not have address limits and can access the expanded map by simply using the actual address.

4.7.5 Offset Calculations

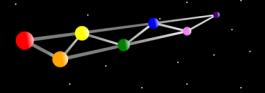
Expanded register map addresses are calculated using the following formula:

```
Register map address = (unit_ID - UID_offset) * UID_factor +  
request_address
```

A register map address offset table is calculated when the configuration is read into memory. Run time overhead consists of just an addition to calculate the address.

Addresses are calculated in a manner such that the highest configurable register map address offset will not be more than the maximum register address minus 65,535.

If the UID factor is large enough, there will not be enough register map addresses available to use all the available UIDs. In this case, any higher UID numbers are considered invalid. For example, if a UID factor of 65,535 was used, UIDs 0 to 15 would fit exactly in the available space, and any higher UIDs could not be used.



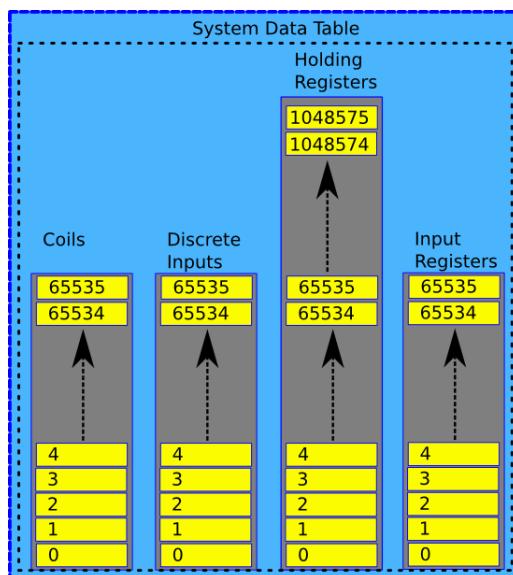
4.7.6 Error Handling

Because of the flexibility of the expanded register map addresses, it is possible for a client to request an address that is out of range or is not configured. These are treated as invalid addresses and an error will be returned to that effect (for Modbus, this is exception 2). These errors fall into the following classes:

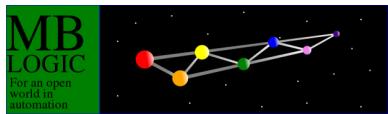
- The message address plus the calculated offset exceed the maximum address in the server register map.
- The unit id is less than the UID offset.
- The unit id is greater than largest offset index that will fit in the available register map space.

4.7.7 Example Using Modbus/TCP Server Protocol

An example of this is the Modbus/TCP server protocol. The addresses transmitted in Modbus messages are 16 bit integers, which limits them to the range from 0 to 65,535.



However, Modbus/TCP also includes a "unit ID" (UID) parameter which is an 8 bit (0 to 255) value. This is normally not used, and the Modbus/TCP server would simply



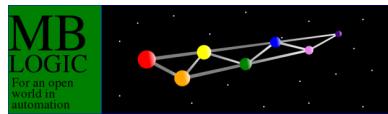
ignore it (it is normally used in Ethernet to serial gateways, to bridge RS-485 or RS-232 systems to Ethernet).

By using the UID as an address offset, the addresses in server requests can be offset by any arbitrary amount. This allows the system to make use of the entire address map even if the individual clients themselves cannot.

Offsets are calculated based on an offset "*factor*" and a UID "*offset*". The offset will offset the UID downwards, to allow lower numbered UIDs to be excluded. The resulting offset UID value is then multiplied by the factor and then added to the address in the request message to obtain the data table address.

For example, assume the configured offset was 0, and the factor was 10,000. If a message arrives with a UID of 0 and a request to read the value at data table address 50, the system will perform the following calculation: $(0 - 0) * 10,000 + 50 = 50$. The resulting address is unchanged. However, if the UID is 10, the address will be $(10 - 0) * 10,000 + 50 = 100,050$. Notice that this address is well outside of the normal limits of a 16 bit address range.

If the UID offset was 1, then the previous example would work as follows: $(10 - 1) * 10,000 + 50 = 90,050$.



4.8 Modbus Support

4.8.1 Overview

Both the server and client versions of this protocol support the following Modbus functions (commands):

- 1 - Read coils.
- 2 - Read discrete inputs.
- 3 - Read holding registers.
- 4 - Read input registers.
- 5 - Write single coil.
- 6 - Write single holding register.
- 15 - Write multiple coils.
- 16 - Write multiple holding registers.

Please see the official Modbus specifications for more information on the Modbus protocol to see how these functions work.

4.9 MB-REST

4.9.1 Overview

The MB-REST web service protocol is considered to be obsolete and should not be used. It will eventually be removed and replaced by a simpler and more effective protocol.

The MB-REST web service protocol does not support expanded register maps.

The MB-REST web service protocol provides access to the data table (memory array) using an HTTP (web) protocol which is modelled after the Modbus protocol. This protocol is intended for applications such as "enterprise integration" (e.g. connecting to an MRP/ERP system). If you are not interested in applications of that nature, you can probably skip this section.

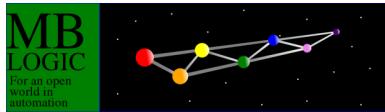
The web service described here is based on REST principles using GET and POST to read and write data. The web service protocol is designed to be similar to the Modbus protocol. If you are familiar with Modbus, this web service should be easy to use.

4.9.2 Web Service Concepts

A true web service (as opposed to just tunneling a remote procedure call over HTTP) treats resources as web pages. This means for example that you can examine the resources using a web browser.

The web service protocol described here treats each data table address as if it were a web page. Since the Modbus function code is inherently part of the address (it determines whether coils, discrete inputs, or registers are being addressed), the function code plus the data table address form part of the address URL. Other parameters, such as quantity, transaction ID, and unit ID are treated as parameters, rather than part of an actual address.

Following normal web conventions, functions which read data and have no side effects are accessed using HTTP GET. Functions which write data or otherwise have



side effects are accessed using HTTP POST. GET and POST data are returned and POST data is sent in XML documents which may be machine parsed.

4.9.3 Reading Data

To read data requires an HTTP GET similar to the following:

```
"http://host/modbus/function-code/address?qty=x&tid=y&uid=z"
```

Where:

- host = host (may include port) name.
- function-code = Modbus function code.
- address = A valid Modbus address.
- x = The quantity of coils or registers to be returned. If "?qty=x" is omitted, the quantity defaults to 1.
- y = The Modbus transaction ID. If omitted, a value of -1 is used.
- z = The Modbus unit ID. If omitted, a value of -1 is used.

4.9.3.1 Supported MB-Rest Function Codes

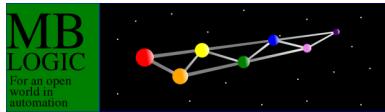
- 1 - Read coils.
- 2 - Read discrete inputs.
- 3 - Read holding registers.
- 4 - Read input registers.

4.9.3.2 Examples

Example 1: This will return 32 coils starting at address 64, with a transaction ID of 67 and a unit ID of 1.

```
http://localhost:8080/modbus/1/64?qty=32&tid=67&uid=1
```

Example 2: This will return 18 holding registers starting at address 0.



```
http://localhost:8080/modbus/3/0?qty=18&tid=68&uid=1
```

Example 3: This will return 1 input register at address 32000.

```
http://localhost:8080/modbus/4/32000
```

4.9.4 Writing Data

To write data requires an HTTP POST similar to that used for reading data.

Example:

```
http://host/modbus/function-code/address?qty=x&tid=y&uid=z"
```

In addition, it requires an XML document to be sent containing the data to be written. The document is embedded in the headers labelled as "modbusrest". The document would be in the following form.

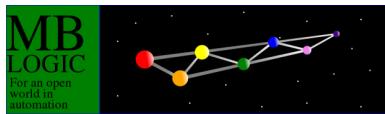
```
<?xml version="1.0" encoding="utf-8" ?>
<request>
    <protocol>modbusrest_v1.0</protocol>
    <msgdata>10010001</msgdata>
</request>
```

The data is contained between the "msgdata" tags (see below for an explanation of data formats).

4.9.4.1 Supported MB-Rest Function Codes

- 5 - Write single coil.
- 6 - Write single holding register.
- 15 - Write multiple coils.
- 16 - Write multiple holding registers.

4.9.5 Server Response



A successful call will return an XML document in one of the following formats:

Example 1 - Response to a successful request to read data using function 1:

```
<?xml version="1.0" encoding="utf-8" ?>
<response status="ok">
    <transactionid>67</transactionid>
    <protocol>modbusrest_v1.0</protocol>
    <unitid>1</unitid>
    <functioncode>1</functioncode>
    <msgdata>10010001</msgdata>
</response>
```

Example 2 - Response to a successful request to read data using function 3:

```
<?xml version="1.0" encoding="utf-8" ?>
<response status="ok">
    <transactionid>68</transactionid>
    <protocol>modbusrest_v1.0</protocol>
    <unitid>1</unitid>
    <functioncode>3</functioncode>
    <msgdata>f12d001a</msgdata>
</response>
```

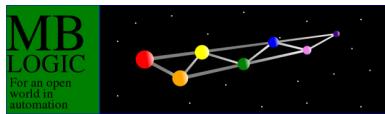
The "function" value will be an echo of the function code sent as in a normal Modbus response message. The "transactionid" and "unitid" will also be and echo of the values sent, or else the default values if these were omitted in the request. "protocol" identifies the protocol name and version.

The "msgdata" will be the returned data (see below for an explanation of data formats). For read functions, this will be the requested data. For write functions, this will be the quantity of elements written (or for function 5, an echo of the data sent).

A failed call will return an XML document in the following format:

```
<?xml version="1.0" encoding="utf-8" ?>
<response status="fail">
    <transactionid>68</transactionid>
    <protocol>modbusrest_v1.0</protocol>
    <unitid>1</unitid>
    <functioncode>133</functioncode>
    <msgdata>3</msgdata>
</response>
```

The "function" and "msgdata" values will normally follow the Modbus protocol rules for exceptions for error code and exception respectively. However, where no



comparable Modbus exception code may apply, the function and exception codes will both be returned as "0". Also however, see the sections on "HTTP Errors" and "Modbus Exceptions".

4.9.6 XML Document Tags

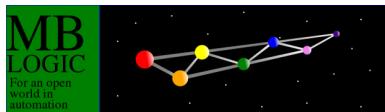
The XML document tags are defined as follows:

- response status = Encloses a server response. It will contain a status indicator of 'status="ok"' or 'status="fail"'. This is not found in client requests.
 - request = Encloses a client request. This is not found in server responses.
 - transactionid = Modbus transaction ID.
 - protocol = Protocol name and version.
 - unitid = Modbus unit ID.
 - functioncode = For replies to successful requests, this is an echo of the Modbus function code sent. For replies to failed requests, this is the Modbus error code.
 - msgdata = For replies to successful requests, this is the Modbus data (see below for format). For replies to failed requests, this is the Modbus exception code.
-

4.9.7 Data Formats

For coil or discrete input data is represented as a string of "0" and "1" characters. The first (left most) character is the value at the requested address, with additional values continuing to the right. As in a normal Modbus response, the number of values returned will be a multiple of 8 with the message padded out with extra "0" characters as necessary. Example: 10010001

For register data, the characters are in groups of 4, with each group representing one register. The right most character in each group is the least significant digit. The first (left most) group is the value at the requested address, with additional register values continuing to the right. Example: f12d001a



For function 5 (write single coil), the coil data is represented as "0000" (for "off"), or "FF00" (for "on") as with normal Modbus.

For hexadecimal data, characters may be represented as either upper or lower case.

4.9.8 HTTP Errors

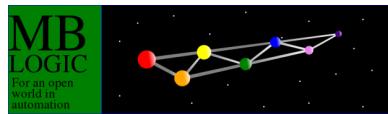
For HTTP errors (as opposed to Modbus exceptions), a standard HTTP error code will be returned (e.g. 404). When an HTTP error code is returned, there will be no XML document accompanying it. There may be a descriptive HTML document, but this is for human consumption only and should not be parsed for error information. The contents of this HTML document may be locale dependent.

4.9.9 Modbus Exceptions

The regular Modbus protocol defines certain errors as being "exceptions". This web service protocol deals with exceptions in the following manner:

- Unsupported function code (exception 1). This is treated as an HTTP error and will return a 404 error (page not found).
 - Invalid address, or address plus quantity is out of range (exception 2). This is treated as an HTTP error and will return a 404 error (page not found).
 - Quantity of inputs or outputs is incorrect, or the data to write is not in the correct format (exception 3). This is treated as a Modbus exception. The HTTP return code will be 200 (OK), and an XML document containing the exception will be returned with it.
 - Internal error reading inputs or writing outputs (exception 4). This is treated as a Modbus exception. The HTTP return code will be 200 (OK), and an XML document containing the exception will be returned with it.
-

4.9.10 Limits on Message Size



The protocol follows the normal Modbus limits on the number of coils, discrete inputs, or registers which may be sent or received in a single request or response. These limits for a single message are:

- Read a maximum of 2000 coils or discrete inputs.
 - Read a maximum of 125 registers.
 - Write a maximum of 1968 coils or discrete inputs.
 - Write a maximum of 123 registers.
-

4.9.11 Protocol Speed and Overhead

Like all web service protocols, this one is slower and has a higher overhead than comparable binary protocols. For example, the regular Modbus/TCP protocol is approximately 10 times faster.

This protocol is best suited to applications which do not require continuous high speed polling, such as interfacing to ERP or production reporting systems.



4.10 HMI Protocol

4.10.1 Overview

The HMI protocol is intended to support a web based HMI system. It supports:

- Tag based data table access.
- Automatic data conversion and scaling.
- Events.
- Alarms.

The HMI protocol is based on JSON. While it is primarily intended for HMI systems, it can also be used as a general purpose web service for other applications such as enterprise integration.

The protocol itself is not described in detail here. Please consult the help pages for the HMI system for more details. The following provides a brief overview.

The HMI system uses a protocol called "Cascadas" which was developed for the purpose.

4.10.2 HMI Servers and Services

The HMI protocol can be configured to provide multiple services. These are:

- **HMI (Standard)** - This is the standard HMI service and includes an integrated web server for loading HMI web pages. It allows read/write access via the HMI protocol.
- **Restricted HMI** - This provides access to the same HMI protocol as the standard version, except that access is read-only. This can be used to provide HMI access to users without providing them the ability to write to the system data table or acknowledge alarms.
- **RSS Monitoring** - This provides an RSS feed of selected HMI events. This is available on both the standard and restricted HMI ports.



- **ERP Protocol** - This is a limited version of the HMI protocol intended to allow access for ERP/MRP applications. The ERP version however limits the features available, and allows tag access to be limited.
- **Alarm and Event Database Access** - The alarm and event database can be queried via a web service API. This does not use the HMI protocol, but is accessed via the HMI servers (both normal and restricted versions).

The normal and restricted HMI servers will use different IP ports. Any available port may be used for either. Both the normal and restricted HMI servers use the same HMI configuration.

4.10.3 Address Tags

Address tags are aliases for addresses in the system data table. They can be used when reading from or writing to individual addresses (or series of addresses) when using the HMI protocol. For example, the tag name "PB1" may be used to represent the system data table coil address 100.

4.10.3.1 Type Conversions

Different data types may be associated with a tag, including integers, floating point numbers and strings. Details of these may be found in the section on "HMI Configuration". However, it is suffice to say here that the HMI configuration system can be used to automatically convert between different data types. For example, numbers can be converted between integers and floating point, depending on what is required for the application.

4.10.3.2 Scaling

Numerical values can be automatically scaled to convert between raw analogue values and engineering units. For example, a 12 bit analogue number (0 to 4095) representing an LVDT reading could be scaled to read the equivalent value in millimeters. The same applies when converting values input from an HMI screen and being written back to the system data table. This scaling can be specified in the configuration and requires no special logic to be written to support it.

4.10.3.3 Reserved Tags

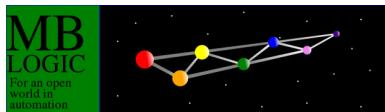
The protocol has several pre-defined "reserved" tags. These are:

- **timeutc** - This is the current time as UTC. This is a floating point number representing a POSIX time stamp.
 - **timelocal** - This is the current time as local time on the server. This is also a POSIX time stamp.
 - **clientversion** - This is a string representing the current version of the *client*.
 - **protocolversion** - This is a string representing the version of the HMI protocol supported by the server.
-

4.10.4 Alarms

Alarms are messages which are brought to the attention of the operator and which the operator must acknowledge. Each alarm is associated with an "alarm tag". Each alarm tag is in turn associated with a coil address in the system data table. The HMI protocol provides alarm messages with the following features:

- **Alarm Tag** - The alarm message passes the alarm tag to the HMI client (e.g. web page), not the actual alarm message. Associating the alarm tag with an actual text message is the responsibility of the HMI client. This means it is possible for the same HMI message to support multiple languages (e.g. both English and French) simultaneously.
- **Time** - The time at which the alarm condition was detected is included in the message. This time is sent as UTC (GMT) so the system can operate independently of time zones.
- **Time OK** - The time at which the alarm condition became "OK" (the cause is no longer present) is also part of the message.
- **State** - The state of the alarm is also included (see below for an explanation of alarm states).
- **Count** - The number of times that an alarm occurred before "going away" is included. This means that if an alarm condition occurs multiple times before being corrected, the number of times this happens can be displayed to the operator. This allows alarms to be summarised in a compact and easy to understand form.



4.10.4.1 Acknowledging Alarms

Alarms must be **acknowledged** by the operator. Acknowledging an alarms means sending a message that the operator has been made aware of the alarm. This is accomplished by sending a message from the HMI client to the HMI server.

4.10.4.2 Alarm States

Alarms can be in the following states:

- **alarm** - The alarm condition is true, and the alarm has not been acknowledged.
- **ackalarm** - The alarm condition is true, and the alarm has been acknowledged.
- **ok** - The alarm condition is false, but the alarm has not been acknowledged yet.
- **ackok** - The alarm condition is false, and the alarm has been acknowledged.
- **inactive** - The alarm condition is false, and the alarm has been acknowledged.

4.10.4.3 Alarm History

Once an alarm message has been acknowledged and the condition causing the alarm is no longer present, the alarm message itself goes into the "alarm history". The alarm history is available for viewing by the operator. The alarm history also includes which operator station acknowledged the alarm.

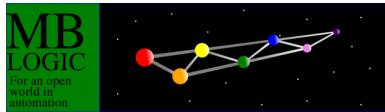
4.10.4.4 Triggering Alarm

Once an alarm has been defined in the HMI configuration it can be triggered by turning on the associated coil address in the system data table. The alarm address may be turned on by any source able to write to the system data table.

An alarm is present when the alarm address is **on** (or "1", or "true").

4.10.5 Events

Events are similar to alarms in that they represent occurrences which are brought to the attention of the operator. Unlike alarms, events do not require the operator to



acknowledge them. While alarms normally represent problems which require the operator to take corrective actions, events are usually simply status information which the operator can review. Like alarms, events are associated with coil addresses in the system data table. Event messages include the following:

- **Event Tag** - Like alarm messages, event messages include an "event tag" representing the event, but not the actual event message text. Again, the reason for this is to allow the HMI system to support multiple languages simultaneously.
- **Time** - This is the time at which the event occurred (in UTC).

4.10.5.1 Triggering Events

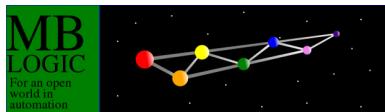
Once an event has been defined in the HMI configuration it can be triggered by turning on the associated coil address in the system data table. The event address may be turned on by any source able to write to the system data table.

Events are "edge triggered". That is, they are triggered when the event address changes from *off* to *on* (or "0" to "1", or "false" to "true").

On start up, the system initialises the event system with the current state of the event addresses. That means that events addresses which are initially "on" will not trigger an event until the event address turns "off" and then "on" again.

4.10.6 Alarm and Event Zones

Alarms, alarm history, and events are grouped into user defined "zones". These zones allow an individual HMI client to ask for and receive only those alarms or events of interest to it. Typical small to medium size HMI systems would normally simply group all alarms and events into a single set of zones (zone definitions for alarms and events are independent of each other). Larger HMI applications however may wish to use several different zones to represent different areas of a machine, production line, or plant.



4.10.7 Alarm and Event Serial Numbers

The system uses message serial numbers to track messages. The actual values used for these serial numbers is not significant and should not be relied on for any other purposes.

4.10.8 Supporting Multiple HMI Systems

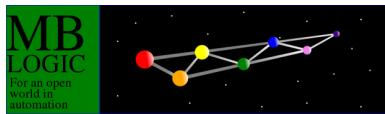
The HMI protocol is designed to inherently support multiple HMI displays without requiring any explicit coordination between them. This includes data, alarms, events, and alarm acknowledge messages.

4.10.9 Logging Alarm History and Events

Alarm history and events are automatically logged to a database. On start up, the system will retrieve the most recent set of alarm history and event messages from the database and make them available for an HMI client.

Current alarms are **not** logged in the database as they represent a dynamically changing condition. Once the alarm has become part of the alarm history however (the alarm condition is no longer present, and the operator has acknowledged it), it is automatically logged as part of the alarm history.

The contents of the database can be retrieved by other programs via a web service protocol. This protocol is documented as one of the system services.



4.11 ERP Protocol

4.11.1 Overview

The ERP protocol is a subset of the HMI protocol which is intended to allow access for ERP/MRP (Enterprise/Manufacturing Resource Planning) applications. This protocol uses the same configuration as the conventional HMI protocol, but limits the available features, and also allows the number of tags and the type of access (read or write).

Like the HMI server, the ERP server can automatically convert types, perform scaling, and enforce data ranges.

4.11.2 Web Service Protocols

The ERP protocol is a type of "web service protocol". A web service protocol operates over HTTP like a web page, but transfers machine readable data rather than a simple web page.

4.11.3 Basic Message Format

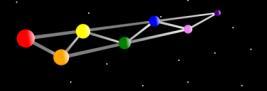
The ERP Protocol uses a subset of the HMI protocol using JSON over HTTP. JSON is a standard encoding of data for web services based on "keys" and "values".

```
{"key1":123, "somekey":"value", "arrayexample":[1, 2, 3]}
```

4.11.3.1 *Basic Message Request Format*

The basic message format consists of the following:

- **id** - This is a client ID string which is used to identify the client. The contents can be any string.
- **msgid** - This is a sequential integer which is used by the client to track requests and responses. The server will echo this number back.



```
{
  "id": "ERP Client 123.",
  "msgid": 456,
}
```

4.11.3.2 Basic Message Response Format

The basic response format contains the following:

- **id** - This is the server ID string which as defined in the HMI configuration.
- **msgid** - This is an echo of the msgid value sent by the client.
- **stat** - This is a string representing a status or error code.

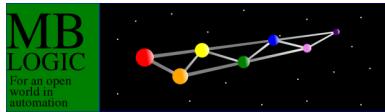
```
{
  "id": "HMIServer demo server",
  "msgid": 456,
  "stat": "ok",
}
```

4.11.3.3 Status Codes

- **unauthorized** - The client is not authorized to communicate with the server.
- **protocolerror** - An error was encountered in the protocol and it could not be accepted. The entire message was discarded by the server.
- **commanderror** - A request command field provided incorrect or invalid data. Check each possible error response for any errors. Multiple errors are possible.
- **servererror** - An unspecified error has occurred in the server which prevents the request from being completed. The error is not specified in any of the return fields.
- **ok** - No errors.

4.11.4 Read Command

The **read** command specifies which address tags should be read. It consists of a list of the address tag names which the client wishes to read. This command is optional and can be left out if it is not needed.



4.11.4.1 Request

```
{  
  "id": "ERP Client 123.",  
  "msgid": 456,  
  "read" : ["Tank1Level", "WidgetCount"],  
}
```

4.11.4.2 Response

```
{  
  "id": "HMIServer demo server",  
  "msgid": 0,  
  "stat": "ok",  
  "timestamp": 1238156675.0,  
  "read": {"Tank1Level": 9875, "WidgetCount" : 87},  
}
```

4.11.5 Write Command

The **write** command is a group of address tags and values which are to be written to the server.

4.11.5.1 Request

```
{  
  "id": "ERP Client 123.",  
  "msgid": 456,  
  "write" : {"Pump1Speed" : 2250},  
}
```

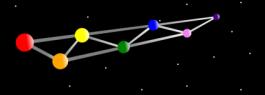
4.11.5.2 Response

```
{  
  "id": "ERP Client 123.",  
  "msgid": 456,  
}
```

4.11.6 Combined Read/Write

Reads and writes can be combined in a single transaction.

4.11.6.1 Request



```
{
  "id": "ERP Client 123.",
  "msgid": 456,
  "read": ["Tank1Level", "WidgetCount"],
  "write": {"Pump1Speed": 2250},
}
```

4.11.6.2 Response

```
{
  "id": "HMIServer demo server",
  "msgid": 0,
  "stat": "ok",
  "timestamp": 1238156675.0,
  "read": {"Tank1Level": 9875, "WidgetCount": 87},
}
```

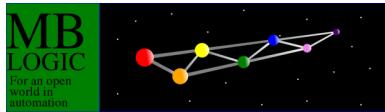
4.11.7 Read and Write Errors

Read and write errors are reported in the "readerr" and "writeerr" fields in the response. The "readerr" and "writeerr" fields are only present in the response if there is a corresponding error to report.

```
{
  "id": "HMIServer demo server",
  "msgid": 9875,
  "stat": "commanderror",
  "timestamp": 1238156675.0,
  "writeerr": {"PBBad": "tagnotfound"}
}
```

4.11.7.1 Error codes

- **accessdenied** - The client does not have authorization to access this tag.
- **tagnotfound** - The address tag is not recognized by the server.
- **writeprotected** - An attempt was made to write to an address which is write protected or otherwise not writable.
- **badtype** - The data value is of an incompatible type and cannot be converted to the desired type.



- **outofrange** - The data value is out of range. Range limits may be imposed by the server based on data type or through configuration limits imposed on a per-tag basis.
 - **addresserror** - An error occurred in attempting to map the tag to the internal server address representation.
 - **servererror** - An unspecified error has occurred in the server which prevents the request from being completed.
-

4.11.8 Configuration

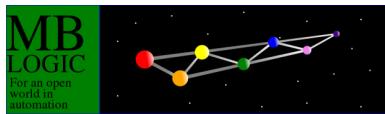
The ERP server uses the same tag configuration as the HMI server. However, the ERP server has the following limits on it:

1. It only supports the "read" and "write" commands. None of the other commands (e.g. alarms, events, etc.) are supported.
2. Only those tags which are specified as being available to the ERP server can be accessed by it.
3. The tags which are configured as being available to the ERP server can be designated as read or write (or both). Tags which are designated as "read" but are not designated as "write" cannot be written to.
4. Conversely, tags which are not designated as "read" cannot be read even if they can be written to (marked "write"). A tag is read-write if the ERP configuration lists it as both "read" and "write".
5. If the underlying memory type is not writable, list a tag as "write" for ERP purposes will not make the tag writable, and you will still receive an error when you attempt to write to it.

Configuration syntax can be found in the section on HMI configuration, and also in the section on using the status web interface for form based configuration.

4.11.9 Creating an ERP Client Interface

To create a custom client interface to the ERP server, you need the following:



- A JSON library. These are available for almost all languages and are standard in many.
- The ability to make an HTTP POST request to a URL. This is also a very common feature for modern programming languages

You then need to do the following:

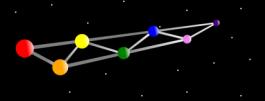
1. Find which server "tags", and make any additions or changes to them which are necessary. This is part of the server HMI configuration.
2. Add the tags to the ERP configuration in the server HMI configuration file. To make the changes active, you may need to reload the server configuration (this can be done while the system is running).
3. Construct the request message and encode it as JSON. You can combine **read** and **write** commands in the same message.
4. Make an HTTP POST request to the server, using the appropriate URL. The server port is configurable. The JSON message should be attached to the request as content.
5. The server will respond with the data attached as content.
6. Decode the response JSON.
7. Check to see if the response "stat" is "ok". If it is not, an error is present.
8. You may check the "msgid" to see if it matches the version sent in the request.
9. If an error was present ("stat" was not "ok"), check the "readerr" and "writeerr" fields for the reason. Errors are normally caused by attempting to use a tag which does not exist or which is write protected, or by attempting to write a value which is of an incompatible type or out of range. The server can be configured to enforce data ranges on a per-tag basis.

4.11.10 Example Client

The following is an example ERP client written in Python.

```
#!/usr/bin/python
# This shows a basic demo for an ERP client.
# This will work with the standard HMI demo.
# 29-Dec-2010.

import json
import urllib2
```



```

# This is the data we are going to send to the server.
# We need a client id, and a message id. In addition, if we want to
# read data, we need to include a list of the tags we want to read. If
we want to
# write data, we need to include a dictionary with the tags and values
that we
# want to write.
basicdemo = {
    'id' : 'ErpDemo',
    'msgid' : 0,
    'read' : ['PL1', 'PL4', 'PumpSpeedActual', 'PumpSpeedCmd',
'Tank1Level',
                    'Tank2Number', 'hello', 'ButtonDisPB'],
    'write' : {'PB1' : 1, 'PumpSpeedCmd' : -2, 'writebad' : 777,
'SSPPGrip' : 1}
}

# Encode the dictionary into JSON format using the standard Python
library.
JsonOut = json.dumps(basicdemo)

# Send the data using POST.
req = urllib2.Request(url = 'http://localhost:8084/', data = JsonOut)
f = urllib2.urlopen(req)

# Read the response.
JsonIn = f.read()

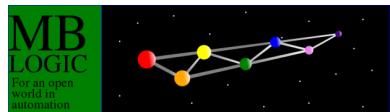
# Convert from JSON into a dictionary.
respdata = json.loads(JsonIn)

# We will iterate through the 'read' response and show the values that
we read.
# The dictionary will also contain other data (such as errors) which
are not
# showing here.
print('Read response was')
readdata = respdata.get('read', {})
for tag, value in readdata.items():
    print('\tTag: %s Value: %s' % (tag, value))

# Read errors.
readerr = respdata.get('readerr', {})
if readerr:
    print('\nRead errors')
    for tag, value in readerr.items():
        print('\tTag: %s Value: %s' % (tag, value))

# Write errors.
writeerr = respdata.get('writeerr', {})
if writeerr:
    print('\nWrite errors')
    for tag, value in writeerr.items():
        print('\tTag: %s Value: %s' % (tag, value))

```

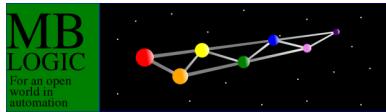


The output from the above is (when used with the standard HMI demo configuration):

```
Read response was
    Tag: PumpSpeedActual  Value: 5
    Tag: PL4   Value: 0
    Tag: PumpSpeedCmd   Value: -2
    Tag: PL1   Value: 1
    Tag: Tank2Number   Value: 75.0
    Tag: Tank1Level   Value: 70

Read errors
    Tag: ButtonDisPB  Value: accessdenied
    Tag: hello   Value: tagnotfound

Write errors
    Tag: SSPPGrip  Value: accessdenied
    Tag: writebad  Value: tagnotfound
```



4.12 Generic Clients

4.12.1 Overview

"Generic clients" are external add-on modules which can be used to implement additional functionality. The main purpose for having generic clients is to make it easier to write serial (RS-232) client interfaces. However, generic clients can also be used for Ethernet communications, file or database access, or any other application where it is desired to add custom functionality.

Generic clients are experimental at this stage, and the interface is subject to change. If you intend to make use of this feature at this stage in its development, be prepared to make minor modifications to your custom generic clients in future versions.

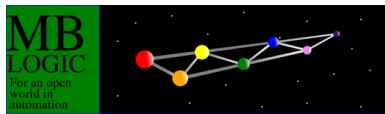
4.12.2 Generic Client Server Interface

Generic clients associate with the main system by communicating with it via the "generic client" server interface. This uses a remote procedure call mechanism called "Perspective Broker" which is part of the Twisted communications framework.

A generic client runs as a separate program that communicates with the main system through the generic server interface. The generic server interface must be configured and running before any generic clients may work with the main system.

4.12.2.1 *Generic Client Names*

Each *instance* of a generic client must be given a unique name. That means that even when running two or more copies of the same type of client, each copy must have its own name. The *name* of the client does not necessarily mean the name of the file containing the client. Two or more clients may use the same code from the same file, but be given different names. The name is simply a parameter passed to the client which it uses to identify itself to the server when it starts up.



For example, the system may need to interface with two identical weigh scales. The same client program file can be used for both, but they must be given different names when they are started so the correct parameters can be passed to them.

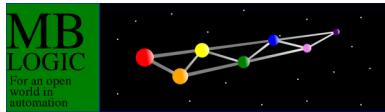
4.12.2.2 Generic Client Parameters

The parameters for each generic client instance are defined in the client communications configuration file ("mbclient.config") just like all other client configurations. These parameters follow a similar format to those used with TCP clients. In addition, each client can have custom parameters which are not "understood" by the server, but are simply passed on to the client for its own use. This allows the standard configuration system to load and parse the parameters while allowing each type of client to have whatever parameters are appropriate to the application.

Details of the configuration parameters can be found in the section on communications configuration.

4.12.3 Generic Client Start and Stop Sequence

1. When the system starts up, it reads and parses the communications configuration file
2. If any generic clients are found, it attempts to start them.
3. Each generic client contacts the generic client interface server and requests its parameter set by name.
4. The generic server interface server sends the parameter set to the client.
5. The client validates its parameters, and sends any error messages back to the server.
6. The generic client runs on a regular scan cycle, performing its own tasks and exchanging data and status information with the server.
7. When the main system wishes to shut down, it sends a "stop" command to each generic client, requesting that they stop executing and exit.
8. The system then waits for a fixed time. When this time delay has passed, the system will forcibly terminate any generic clients which may still be running.
9. The system then completes its own shutdown and exits.



4.12.4 File Locations

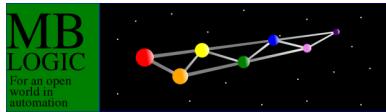
Generic clients are located in a directory called "genclient". The system will automatically look in that directory for any specified files.

4.12.5 Protocols and Message Transports

Generic clients are primarily intended to allow the addition of protocols and custom interfaces. These typically include serial protocols, but can also include protocols that use HTTP and low level socket access. Since the generic client runs independently from the main system, it will not block or interrupt the operation of the main system. However, the client should not block (stop) execution of the main client loop for long periods of time as the main system may interpret that behaviour as indicating the client has faulted ("locked-up").

Some available libraries for implementing client protocols are:

- `socket` - This is a standard Python library that offers direct TCP, UDP, and unix socket access.
- `urllib2` - This is a standard Python library that offers simple HTTP protocol access.
- `pyserial` - This is a third party library that offers access to the serial port.



4.13 Generic Client Framework

4.13.1 Overview

This section describes the "generic client framework". This is a library which is used to help create custom generic clients. The generic client framework handles communications between the generic client and the main system. It also operates the "scan" cycle which calls the user protocol code on a regular schedule.

Generic clients are experimental at this stage, and the interface is subject to change. If you intend to make use of this feature at this stage in its development, be prepared to make minor modifications to your custom generic clients in future versions.

4.13.2 Framework API

The generic framework library is located in a file called "**GenClientLib.py**". This file contains the following classes:

- GenClientController - Handles server communications and scan scheduling.
- GetOptions - Optional class for reading command line options.

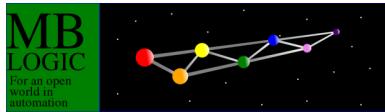
4.13.2.1 *GenClientController*

GenClientController handles communications with the generic interface server, and scheduling of the user protocol code scan.

```
# Initialise the generic client handler.  
gencontrol = GenClientLib.GenClientController(port, clientname,  
UserClient)
```

The initialisation parameters are:

- port - The port number of the server. This will have been passed to the client as a command line parameter.



- clientname - The name of the client instance. This will have been passed to the client as a command line parameter.
- UserClient - This is a user written class which implements the actual generic client logic.

4.13.2.2 *StatCodes*

StatCodes formats command status messages. It expects the following parameters:

- cmd (string) - A command name.
- statcode (string) - A command status code.

It returns a tuple containing a command status message in the format expected by the status reporting system on the server.

4.13.2.3 *GetOptions*

GetOptions reads and analyses the command line parameters which are set by the system when it starts a client. These parameters are:

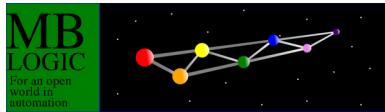
- p - The port number.
- c - The client name.
- d - The start delay time (in seconds).

The class contains one method ("GetStartParams") which returns the port, client name, and delay.

```
# Get the command line parameter options.  
CmdOpts = GenClientLib.GetOptions()  
port, clientname, startdelay = CmdOpts.GetStartParams()
```

"GetStartParams" is a convenience class and is not an essential part of the generic client framework. Alternate implementations may be used if a generic client requires more command line parameters.

4.13.3 User Protocol Class



The generic client framework expects a class which implements the actual client communications. This class must include the following methods:

- GetClientMsgs - This returns the client messages.
- GetStatus - This returns the status.
- SetParams - This is used to set the configuration parameters.
- NextCommand - This is called each scan to run the main client code loop.

Writing user protocol classes is discussed in another section.

4.13.4 Signal Handlers

If you wish to be able to shut down the user client gracefully using a keyboard interrupt, you will need to include signal handlers.

```
# Signal handler.
def SigHandler(signum, frame):
    print('\nOperator terminated generic client %s at %s' % (clientname,
time.ctime()))
    gencontrol.StopClient()

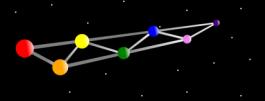
def SigTermHandler(signum, frame):
    print('\Received terminate signal for generic client %s at %s' %
(clientname, time.ctime()))
    gencontrol.StopClient()

# Initialise the signal handler.
signal.signal(signal.SIGINT, SigHandler)
signal.signal(signal.SIGTERM, SigTermHandler)
```

4.13.5 Example

The following shows a simple example with the actual protocol implementation left out.

```
#!/usr/bin/python
```



```

"""
This file should contain the user routines used to help implement the
protocol
    specific generic client functions.
"""

# These are some example imports, which may not be required in another
implementation.
import time
import urllib2
import signal
import json

# This import handles the generic client framework.
import GenClientLib

#####
class UserClient:
    """
    """

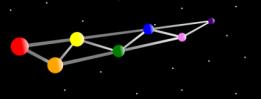
    #####
    def __init__(self):
        """The initialisation code goes here.
        """
        # This formats the command status messages
        self._StatusMsgs = GenClientLib.StatCodes()

    #####
    def GetClientMsgs(self):
        """Return the list of client messages.
        """
        return self._ClientMsgs

    #####
    def GetStatus(self):
        """Return the current status.
        """
        return self._ConnectStatus, cmdstat

    #####
    def SetParams(self, hostconfig, clientconfig, cmdlist):
        """Accept the configuration parameters and validate
them.
        """
        # Do something with these parameters ...
        pass
"""


```



```

        def NextCommand(self, readable, servercmd):
            """This should execute the next command, whatever it
is. This
            is called regularly by GenClient.
            """

            data = {}
            # First check if we've got a good parameter set.
            if self._ConfigOK:

                # Various details go here...

                # Set the connection status.
                self._ConnectStatus = 'running'

                nextpoll = 0.3

            else:
                # Set the connection status.
                self._ConnectStatus = 'stopped'

                nextpoll = 1.0

            return data, nextpoll

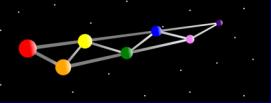
#####
# Signal handler.
def SigHandler(signum, frame):
    print('\nOperator terminated generic client %s at %s' %
(clientname, time.ctime()))
    gencontrol.StopClient()

    def SigTermHandler(signum, frame):
        print('\Received terminate signal for generic client %s at %s' %
(clientname, time.ctime()))
        gencontrol.StopClient()

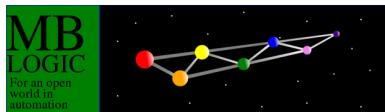
#
# Initialise the signal handler.
signal.signal(signal.SIGINT, SigHandler)
signal.signal(signal.SIGTERM, SigTermHandler)

#####
# Get the command line parameter options.
CmdOpts = GenClientLib.GetOptions()
port, clientname, startdelay = CmdOpts.GetStartParams()

```



```
# Initialise the generic client handler.  
gencontrol = GenClientLib.GenClientController(port, clientname,  
UserClient)  
  
print('\n\nStarting generic client %s at %s' % (clientname,  
time.ctime()))  
  
# Delay the specified number of seconds. This will allow the main  
# program to start up before trying to contact it.  
time.sleep(startdelay)  
  
# Start the generic client.  
gencontrol.StartClient()  
  
# This doesn't get executed until the client halts.  
print('\n\nGeneric client %s halted at %s' % (clientname,  
time.ctime()))
```



4.14 Generic Client User Protocol Class

4.14.1 Overview

This section describes how to write a simple user client protocol class for a generic client. It is assumed you are familiar with writing simple Python programs.

Generic clients are experimental at this stage, and the interface is subject to change. If you intend to make use of this feature at this stage in its development, be prepared to make minor modifications to your custom generic clients in future versions.

4.14.2 Import Modules

"GenClientLib" must be imported to provide communications with the server.

```
import GenClientLib
```

Additional imports can be included to support the client features. This example is

```
import time
import signal

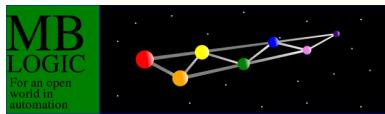
import urllib2
import json
```

4.14.3 Client Protocol Class

4.14.3.1 Create a Class

The client protocol must be implemented as a class. This class may import other classes as needed. This class may be named anything.

```
class UserClient:
```



The generic client framework expects a class which implements the actual client communications. This class must include the following methods:

- `GetClientMsgs` - This returns the client messages.
- `GetStatus` - This returns the status.
- `SetParams` - This is used to set the configuration parameters.
- `NextCommand` - This is called each scan to run the main client code loop.

4.14.3.2 *GetClientMsgs*

`GetClientMsgs` must return a list of strings containing the client messages. Client messages are any messages the generic client wishes to return to the server to be displayed in the communications monitor status interface. These would typically describe errors, but may involve whatever other information was felt appropriate.

```
def GetClientMsgs(self):
    return self._ClientMsgs
```

4.14.3.3 *GetStatus*

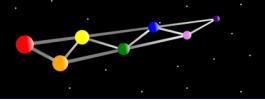
`GetStatus` must return a tuple containing the connection status, and the command status. Connection status is a string containing the connection status code. The connection status code must be one of the following:

A list of the valid status codes may be found in the section on "Communications Errors".

The command status must be a dictionary in the following format: "`{<command name> : (<status number>, <status code>, <status description>)}`"

There should be one dictionary entry per command. These have the following definitions.

- command name - The name of the command.
- status number - An integer numerical code. This value will be written to the system data table fault addresses to indicate the current status. If there are



multiple commands, only one of these codes will be used (see the documentation on fault address configuration for details).

- status code - A short string code used to indicate the status. This should correspond to the status number.
- status description - An optional string providing a more detailed text description of the status. If this is unused, an empty string should be supplied as a placeholder.

A list of the valid command status codes may be found in the section on "Communications Errors".

```
def GetStatus(self):
    cmdstat = self._CmdStatusBuff
    self._CmdStatusBuff = []
    return self._ConnectStatus, cmdstat
```

4.14.3.4 SetParams

This is a method which is called by the generic client framework when a new set of parameters is available. It must accept 3 parameters. There are

- hostconfig - This is a dictionary containing the standard configuration parameters used by the server. The generic client does not typically need to use this information.
- clientconfig - This is a dictionary containing the client parameters. These parameters have not been validated by the server.
- cmdlist - This is a list of commands for the client. These must be validated by the client.

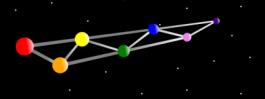
```
def SetParams(self, hostconfig, clientconfig, cmdlist):

    # Validate the expected parameters.
    repeattime = int(clientconfig['repeattime']) / 1000.0
    retrytime = int(clientconfig['retrytime']) / 1000.0

    # Etc. ...

    # Parse the commands.
    # Etc. ...

    # Send a message back.
    self._ClientMsgs.append('The parameters were ok.')
```



4.14.3.5 *NextCommand*

This gets called on a regular basis by the generic client framework. The main generic client code would go here. This must accept two input parameters, and return two parameters. The input parameters are:

- **readtable** - This is a dictionary containing a mirror of the requested portion of the server data table.
- **servercmd** - This is a string containing the server command.

Note: Server commands to the client are not implemented at this time. However this parameter is required to be present for compatibility with future use. Server commands are used to shut down the system, but these are intercepted by the client framework.

Return parameters:

- A dictionary in the same format as the "readtable" input parameter. This will contain the data to be written to the server.
- The time in seconds to wait before running this function again.

```
def NextCommand(self, readtable, servercmd):
```

The data format used for the data table transfer parameters is as follows. They consist of a dictionary with keys that match the names used in the communications configuration file ("coil", "inp", "inpreg", "holdingreg"). Each dictionary value is a list containing the data values for the data table. Coils and discrete inputs are boolean values, and input and holding registers are signed 16 bit integers.

Unused types may be either an empty list, or the key/value pair may be absent from the dictionary. Generic client writers must not depend on any particular key/value pair to be present if it is unused.

```
{'coil' : [True, False, True, True], 'inp' : [True, False, True, True],
```



```
'inpreg' : [], 'holdingreg' : [1234, 5678]}
```

The following shows a simple example which uses the HMI protocol to read data from the server data table. The list of HMI "tags" to read was passed as part of the command. This uses the standard Python "urllib2" library to send and receive the messages (over HTTP).

```
def NextCommand(self, readable, servercmd):
    data = {}
    # First check if we've got a good parameter set.
    if self._ConfigOK:

        # Get the next command.
        try:
            cmdname, cmdvalue = self._CommandIter.next()
            nextpoll = self._CommandTime
        except StopIteration:
            self._CommandIter = iter(self._CommandList)
            cmdname, cmdvalue = self._CommandIter.next()
            nextpoll = self._RepeatTime

        # Increment the message id.
        self._MsgID += 1
        if self._MsgID > 65535:
            self._MsgID = 0

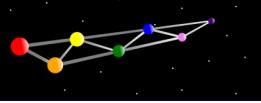
        self._basemessage['msgid'] = self._MsgID
        self._basemessage['read'] = cmdvalue

        # Encode the dictionary into JSON format using the
        # standard Python library.
        JsonOut = json.dumps(self._basemessage)

        # Send the data using POST.
        req = urllib2.Request(url = 'http://localhost:8082/',
data = '',
                           headers =
{'Cascadas': JsonOut})
        f = urllib2.urlopen(req)

        # Read the response.
        response = f.read()

        # Get the response data. We look for the blank line
        between the headers and
                           # the body and assume that everything after the first
        blank is the data.
```



```
# This is a feature of HTTP, and not something which is
part of the Cascadas
# protocol itself.
resplist = response.splitlines()
JsonIn = ''.join(resplist)

# Convert from JSON into a dictionary.
respdata = json.loads(JsonIn)

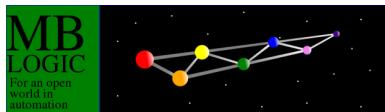
rdata = respdata['read']
coillist = map(lambda x: rdata.get(x, False), ['PL1',
'PL2', 'PL3'])
data['coil'] = coillist
reglist = map(lambda x: rdata.get(x, 0), ['Tank1Level',
'PL4'])
data['holdingreg'] = reglist

# Set the connection status.
self._ConnectStatus = 'running'
# Set the command status.
self._AddCmdStatus(cmdname, 'ok')

else:
    # Set the connection status.
    self._ConnectStatus = 'stopped'

nextpoll = 1.0

return data, nextpoll
```

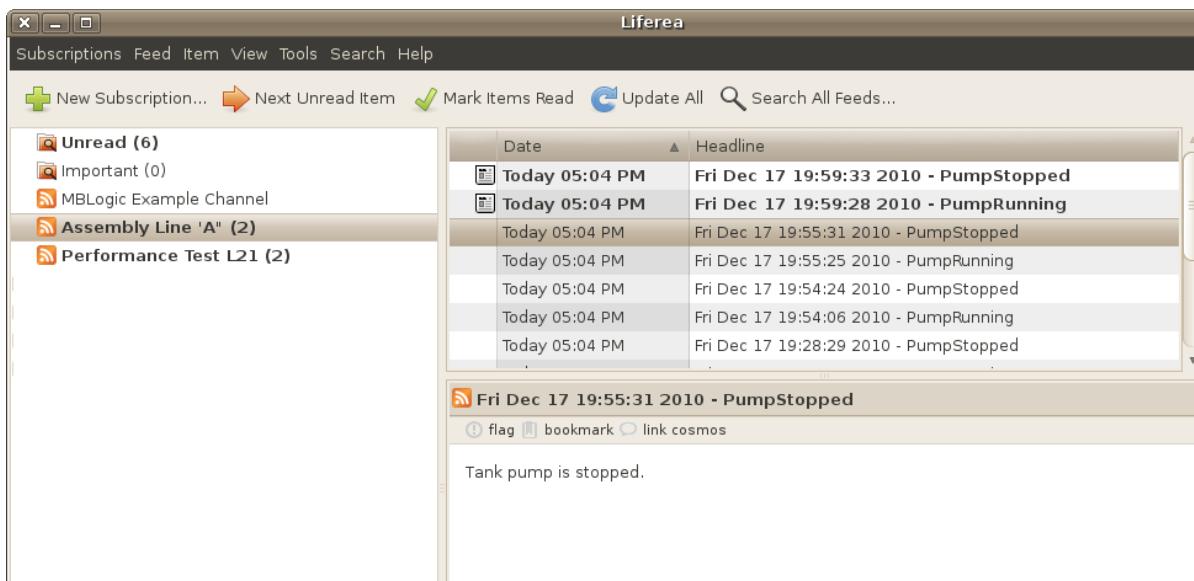


4.15 RSS Feed

4.15.1 Overview

An "RSS Feed" is a means of allowing standard clients to automatically poll a server for information about updated conditions. There are many popular RSS clients available.

The RSS in this system can be used to monitor events. You can configure the RSS server to display selected events as an RSS feed. This allows users to automatically monitor any number of systems without requiring any administration effort.

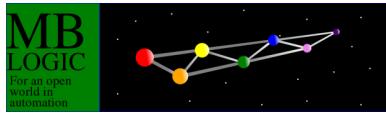


4.15.2 Addressing

The RSS feed is available via the HMI server (both the standard and the "restricted" versions).

```
http://localhost:8082/rss/events.xml
```

4.15.3 Configuration



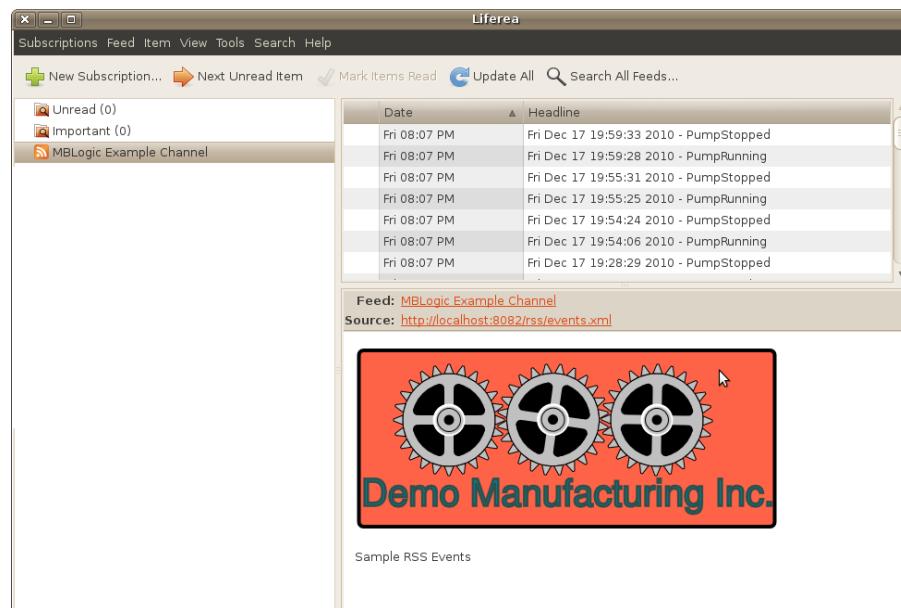
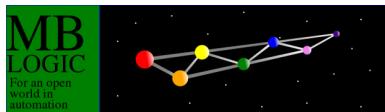
The RSS feed is configured via a template file called "*rsstemplate.xml*". The event message texts are stored in a file called "*rsseventtexts.js*".

4.15.3.1 *rsstemplate.xml*

The RSS template file contains the following items:

- **taglist** - This is a list of the event tags which will be monitored by the RSS system.
- **linkfile** - This is the file which will be displayed if the user clicks on the link in the RSS message.
- **logfile** - This is the default logo file which will be displayed if no individual RSS message is selected.

```
<?xml version="1.0"?>
<rss version="2.0">
    <channel>
        <title>MBLogic Example Channel</title>
        <linkfile>hmidemo.xhtml</linkfile>
        <taglist>PumpRunning,PumpStopped</taglist>
        <logfile>pagelogo.png</logfile>
        <description>Sample RSS Events</description>
        <image>
            <title>RSS Demo</title>
            <url>% (logfile)s</url>
        </image>
        % (messages)s
    </channel>
</rss>
```



4.15.3.2 rsseventtexts.js

The event texts file contains the text messages which are associated with the event tags. The message format is the same as used for the HMI event texts. However, the content of the text messages used for RSS events do not have to be identical to those used in the HMI.

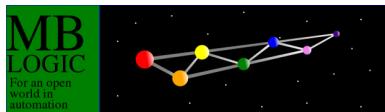
```
{
  "PumpRunning" : "Tank pump is running.",
  "PumpStopped" : "Tank pump is stopped."
}
```

4.15.4 The RSS Template

The RSS template must be customised for each application. The template is a simple XML file that can be edited with any plain text editor. The template includes the following elements.

4.15.4.1 Title

The title is the name which will appear as the name for the RSS feed. Change this to the name you wish to have appear as the title.



```
<title>MBLogic Example Channel</title>
```

4.15.4.2 Description

The description will appear as the default text if an individual message is not selected.

```
<description>Sample RSS Events</description>
```

4.15.4.3 Message

This is a target used by the software to insert the actual RSS messages. Do not remove or alter this.

```
% (messages)s
```

4.15.4.4 Image

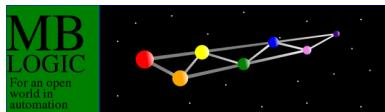
A default image or logo can be specified which will be displayed whenever a specific RSS message is not selected. The text between the "title" XML tags is displayed with the image or logo. Enter whatever text is appropriate. The "%(logfile)s" is used as a target for the software to insert the actual link to the file. Do not remove or alter this.

```
<image>
    <title>RSS Demo</title>
    <url>% (logfile)s</url>
</image>
```

4.15.4.5 Link File

A web page can be specified which will be launched in a web browser whenever the user clicks on a link in the RSS message (this is a feature in typical RSS reader software). For example, the link file could be an HMI page.

Specify only the file name. The system will automatically insert the path, IP address, and port. The link file web page must be located in the HMI directory.



```
<linkfile>hmidemo.xhtml</linkfile>
```

4.15.4.6 Logo File

The logo file is a default image which will be displayed by the RSS reader (if the reader supports that feature). Like the link file, only specify the file name. The system will insert the other information automatically.

```
<logfile>pagelogo.png</logfile>
```

4.15.4.7 Tag List

The tag list is the list of event tags which you wish to monitor via the RSS system. These must be valid event tags, and tag names must be separated by commas.

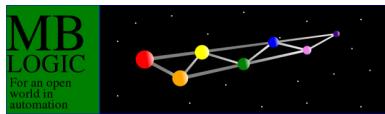
Specifying the event tag list allows just those events which are the most relevant to an RSS feed to be listed. In most cases it is not useful to include too many events in the RSS feed. Rather, just those events which represent significant changes should be monitored. Overwhelming users with too many events would cause the most significant ones to be lost in the flood of minor ones.

```
<>taglist>PumpRunning,PumpStopped</taglist>
```

4.15.5 Event Texts File

The event texts file contains the actual messages which are associated with the event tags. These texts *may* be the same as the ones used in the HMI web page, but do not *have* to be. They may be longer, shorter, or differently worded (the users viewing the RSS feed may need a simpler explanation).

The RSS events text file can be copied from the version used in the HMI web age, but with the following differences:



- It only needs to contain the tag names which are actually used in the RSS template file (in the tag list), and text messages associated with them.
- There must be no Javascript variable name used at the start of the file, and no semi-colon closing it. The first character must be a "{", and the last character "}". The data must be in valid JSON format.

```
{  
    "PumpRunning" : "Tank pump is running.",  
    "PumpStopped" : "Tank pump is stopped."  
}
```

4.15.6 Loading and Changing the RSS Messages

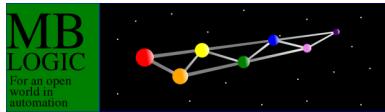
The RSS template and the message texts are automatically loaded when the system starts up. They are also re-loaded whenever the HMI configuration is re-loaded. In other words, to change the RSS messages while the system is running, reload the HMI configuration.

4.15.7 Server Configuration

The RSS message system is part of the HMI server. The same RSS feeds will operate through both the normal and "restricted" HMI servers.

The access path to the RSS feed is the same as would be used to access the equivalent HMI web page except using the path "rss/events.xml" instead of the HMI web page name.

```
http://localhost:8082/rss/events.xml
```



4.16 Modbus Basics

4.16.1 Overview

Modbus is the leading industrial open control protocol. This help page will discuss some of the basic concepts and correct some common misconceptions, but does not provide a detailed description of the protocol itself.

4.16.2 Types of Modbus

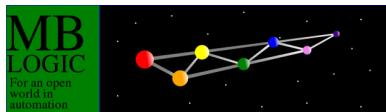
Modbus comes in several different types, depending upon the media over which it is transported.

- Modbus RTU - This is original Modbus. It is used over RS-232 and RS-485 links.
- Modbus ASCII - This is similar to Modbus RTU, but the data is encoded in ASCII instead of raw binary. This version is mainly used over radio links.
- Modbus/TCP - This version is used over Ethernet. It is similar to Modbus RTU, but uses the check-sum built into Ethernet rather than including the RTU check sum.
- Modbus over Ethernet - This is a vague term used by some vendors, but is not an officially recognised name. It is often used to refer to tunnelling Modbus RTU over Ethernet between two points using special hardware. This is **not** part of the Modbus standard and the format is vendor dependent.
- Modbus/UDP - This is similar to Modbus/TCP but uses UDP Ethernet sockets instead of TCP sockets. This is offered by some vendors, but is **not** part of the Modbus standard.
- Modbus+ - This is a proprietary protocol which uses the Modbus name, but doesn't follow the Modbus communications standard. This is rarely encountered.

MBLogic uses Modbus/TCP.

4.16.3 Communications

4.16.3.1 Client/Server or Master/Slave:



Like most communications protocols, Modbus uses a ***client/server*** type protocol. This is also sometimes referred to as ***master/slave***. A "master" is the same thing as a "client", while a "slave" is the same thing as a "server". The master/slave terms originated in industry, while the client/server terms originated in the computer industry. The different terms arose due to historical differences as to how they were being applied, but the computer industry terms have to a large degree replaced the industrial terms as industry has adopted more off the shelf computer technology. Typically, the client is the PLC or controller, while the server is a field device such as a valve bank or sensor block.

A "client" sends a ***request*** to a "server". The server decodes the request and sends back a ***response*** with the requested data or an acknowledgement. This is the same as how other common protocols work.

For example, when you use a web browser to view a web page on the internet, your web browser sends a "page request" to the web server. The web server decodes the request and sends back a web page as a "response". Your e-mail client program fetches your e-mail in the same way from a mail server.

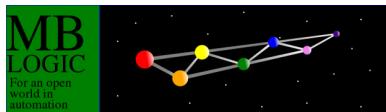
4.16.3.2 Unit ID

A Modbus message includes what is called the ***unit ID***. A unit ID is a number between 0 and 255 which is used to identify the server (or slave) address in RS-232 or RS-485 networks. Each server (slave) is assigned a "slave ID" number and listens for messages which contain this number in the unit ID field.

Modbus/TCP also has the unit ID in its messages, but the Ethernet TCP/IP address is used to decide where to actually delivery the message. Many or most server devices will ignore the unit ID. However, some will use the unit ID to decide whether to forward the message out a built-in serial port. This message forwarding allows older RS-485 devices to be used on newer Ethernet networks. However, support for this feature is only found in a few devices.

4.16.3.3 Message ID

When a Modbus message sends a request, it includes a ***message ID*** number. This is a number from 0 to 65,535. This number is normally incremented by the client for each



request (and allowed to roll over to 0 again when it overflows). This message ID is echoed back by the server. The client can use this message ID number to determine if any messages are being lost or delayed in transmission.

4.16.3.4 *Serial Communications Parameters*

When used over RS-232 or RS-585 networks, the usual serial communications parameters such as baud rate, parity, stop bits, etc. must be set correctly for all devices. Setting of these parameters is not discussed here. Consult your vendor documentation for details.

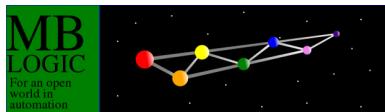
4.16.3.5 *Ethernet Parameters*

When used over Ethernet, Modbus/TCP does not require setting the baud rate or other similar parameters. However it does require the use of the correct ***IP address*** and ***port number***.

An "IP address" is the network address of the device which the message is being sent to. Each device on a network must have a unique IP address. The number of possible IP addresses is very large (billions of addresses).

In addition to the IP address, the "port number" is required. An Ethernet "port number" does not refer to the physical connector. A port number is a number send in the Ethernet header used by the operating system or firmware in a device to determine which program a message should be sent to. This allows multiple programs to share the same Ethernet adaptor without conflict. When a client sends a message, it sends it specifies a particular port number. When the message is received by the server, the operating system or firmware will look at the port number and route the message to the correct program.

When a server program starts up it will "bind to a port". That simply means that it will ask the operating system to be assigned a particular port number. Only one server may bind to the same port number at any time. If the port number is already in use by another program the new server program will not be allowed to use it, and will encounter an error. When a server program shuts down, it frees the use of the port. The operating system will then wait for a certain time-out to pass (the standard is 70 seconds) before allowing that port number to be used again.



The "standard" port number for Modbus/TCP is port 502. So far as the protocol itself is concerned, the port number used by it is completely transparent. The port number is not sent in the Modbus protocol message itself. However, port 502 is recognised by convention as the standard port number to use, so most field devices will be listening on port 502. However, it is possible to run Modbus/TCP on alternate port numbers provided that all devices that are participating in the communications can be configured to use the alternate port.

4.16.4 Modbus Data Representation:

4.16.4.1 Data Table

Modbus uses the concept of a ***data table*** to refer to data. Data tables should be familiar to anyone who has used a PLC. A data table is an array or block of memory used to store data. Data is referenced using data table addresses. Modbus data table addresses come in four types.

- Discrete inputs - These are read only boolean values. They are typically used to represent sensor inputs and other boolean values which are read but not written to by the user.
- Coils - These are read-write boolean values. They are typically used to represent outputs (e.g. valve solenoids) or internal bits which are both read by and written to by the user.
- Input registers - These are read only 16 bit integers. They are typically used to represent analogue input values and other integer values which are read but not written to by the user.
- Holding registers - These are read-write 16 bit integers. They are typically used to represent analogue outputs or internal numbers which are both read by and written to by the user.

In addition to the native data types of boolean and integer, it is possible to store large integers, floating point numbers, and strings in a Modbus data table by splitting the data over several registers. However, Modbus does not offer any direct support for this, so the user is responsible for splitting the values up and storing them in separate locations.



4.16.4.2 Data Table Addresses

Modbus data table addresses are simply integer numbers. Each address type (discrete input, coil, input register, holding register) has its own numbered set of addresses. For example, discrete input 42 and coil 42 are two separate addresses.

Modbus distinguishes between the **protocol** address and the **data model** address. The addresses that are used in the protocol are numbered from 0 to 65,535. However, vendor documentation may number from 1 to 65,536. This documentation will refer to the **protocol** addresses (that is the first address is 0).

Although it is possible to have 65,536 addresses of each type the number of addresses actually implemented in a particular device (sensor block, valve bank, PLC, etc.) is usually much less than this. Each device designer is responsible for deciding what makes sense for their application. They will then normally supply a "memory map" or list of addresses for their device listing what addresses were implemented, and what each one does.

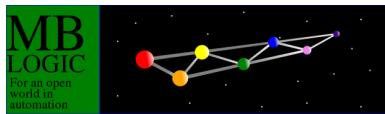
It is also possible to "overlay" address types. That is, it is possible to design a device such that several different address types all refer to the same physical memory location. For example, it is possible to have holding register 10 and input register 10 both be the same memory location. In this case if you were to write to holding register 10, the value would also appear in input register 10. The same applies to coils and discrete inputs.

In the same way, it is possible to pack coils and discrete inputs in registers. For example, coils 0 to 15 could be packed in holding register 0, coils 16 to 31 in holding register 1, etc. This would allow multiple coils to be read (or written to) as words.

Although overlay of data types and packing of coils into registers is possible, it is not often found in practice.

4.16.4.3 Some Common Misconceptions about Modbus Addresses

According to the Modbus standard, addresses are simply integers from 0 to 65,535 with the different address ranges being referred to as coils, holding registers, etc. However, some vendors will document their hardware using numerical prefixes which



are not actually part of the Modbus address. This originated from some models of PLCs which used the Modbus communications protocol, and which also used numerical prefixes in their internal data table. This is similar to using "I", "Q", "V", etc. as address prefixes in IEC type PLCs.

However, it is important to remember that these numerical prefixes are documentation methods and are *not* part of what the Modbus protocol itself sends as part of the messages. A difference in documentation methods does not affect the compatibility of the protocol itself.

These prefixes are they mentioned anywhere in the Modbus standard, but the following shows how they are typically used in documentation based on this older convention:

- 0xxxx - Coils.
- 1xxxx - Discrete inputs.
- 3xxxx - Input registers.
- 4xxxx - Holding registers.

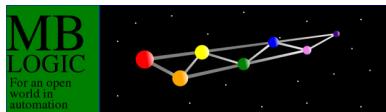
Note that there is no 2xxxx address prefix.

In addition to numerical prefixes, some documentation will refer to protocol addresses (addresses start at 0), while other documentation will refer to data model addresses (addresses start at 1). That is, the first holding register may be 0 or 1 (or 40000 versus 40001 using prefixes). However, this has no bearing on what gets sent over the wire as a Modbus message. For a Modbus protocol message, the lowest address is always "0", not "1".

This document uses standard Modbus terms and addresses throughout using Modbus protocol addressing without prefixes or offsets. However, if you are reading the documentation for sensor blocks, valves, and other devices, you must keep in mind that some vendors may document their hardware in different ways.

4.16.5 Modbus Commands, or "Functions"

4.16.5.1 Functions



Modbus commands are known as *functions*. A function is simply a command to read or write a data table address. Functions are numbers such as 1, 2, 3, 4, etc. For example, function "1" will read one or more coils. Function "15" will write to one or more coils. All function codes are defined as part of the Modbus standard, but which functions were actually implemented in any particular device is up to the device designer. For example, a valve bank may only implement functions for writing coils because that is all that was necessary for that device.

The most common functions are listed below. There are many other functions defined in the Modbus standard, but these are the ones most commonly encountered.

- 1 - Read multiple coils.
- 2 - Read multiple discrete inputs.
- 3 - Read multiple holding registers.
- 4 - Read multiple input registers.
- 5 - Write single coil.
- 6 - Write single holding register.
- 15 - Write multiple coils.
- 16 - Write multiple holding registers.

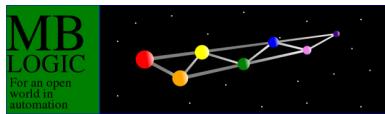
4.16.5.2 *Quantity*

Modbus functions which read or write multiple addresses will also require a *quantity* parameter. The "quantity" parameter specifies the number of consecutive addresses to read or write.

4.16.5.3 *Maximum Number of Addresses Which Can be Read or Written at Once*

The modbus protocol specifies the maximum number of addresses which can be read or written at one time. This limits the amount of data which must be transferred in a single command to no more than 255 bytes. For a read command, the limit is 2000 coils or discrete inputs or 125 registers. For a write command, the limit is 1968 coils or 123 registers.

4.16.6 *Modbus Errors*



4.16.6.1 Modbus Faults

When a Modbus message is received by a server it is analysed to see if all the parameters are correct. If everything is OK, the server will send a response. This response may consist of data, or it may just be an acknowledgement that the message was received.

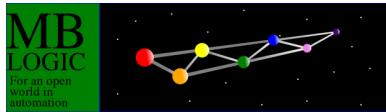
If there are any errors however, the server may do one of two things. If the message could not be decoded at all (e.g. there was a bad check sum), the message is simply discarded. If the message could be decoded but one or more parameters were incorrect (e.g. an attempt was made to read a non-existent address), then the server will send a response containing a ***fault code***. The fault code is simply the function code from the request with 128 added to it.

For example, if the client sends a request with function 3 (read holding registers) that contains an error, the server will respond with a fault code of 131 ($3 + 128 = 131$).

4.16.6.2 Modbus Exceptions

In addition to the fault code, the server will return an ***exception code*** which provides more detail about what was wrong. Exception codes are defined individually for each function, but for most common functions the following will apply:

- 1 = The requested function code (command) is not supported.
- 2 = The address is incorrect (does not exist). For functions which operate on multiple addresses, this check includes all addresses which are affected by the request.
- 3 = The quantity of addresses is incorrect. The quantity parameter was too large (or was zero) for the function requested.
- 4 = Some unspecified error occurred in the server which prevented the request from being carried out.



5 Soft Logic

5.1 Overview

The soft logic system runs as an optional independent subsystem. It is not necessary to configure or program the soft logic system if it is not required.

The soft logic system is designed as a replaceable library. The library installed in this version is the "Ck Logic Engine" (or just "Ck").

5.1.1 Help Topics

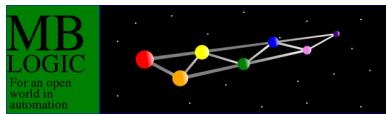
5.1.2 Specifications and Features:

- [Specifications](#) - General specifications.
- [Addressing](#) - Data types, constants, addresses, and subroutine names.
- [Instructions](#) - Instructions.

5.1.2.1 Programming and Configuration

- [Programming](#) - Writing programs.
- [Configuration](#) - Configuring the soft logic system.
- [Program Monitoring and Reloading](#) - Monitoring and reloading the program.
- [Data Table Save/Restore](#) - Saving and restoring the data table.

Also see the "System Status" documentation for information on how to use the web interface to view and edit configurations.



5.2 Specification

5.2.1 Overview

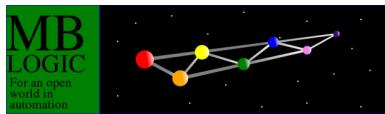
The following are the specifications for the "Ck" soft logic library.

5.2.2 General Specifications

Characteristic	Limits
Data table size	50 kbytes
Digital inputs	2000
Digital outputs	2000
Analogue inputs	125 ^{1, 2}
Analogue outputs	125 ^{1, 2}
Timers	500
Counters	250
Maximum number of instructions	No limit ³
Maximum number of subroutines	No limit ³

¹Plus an equal number of signed addresses.

²In addition to these addresses, any register type can be used for I/O, not just the "analogue" addresses.



³There is no enforced limit to the size of program that can be run. The practical limitations are related to the resulting scan rate and the time to load the program on start up.

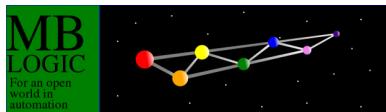
5.2.3 Instruction Speed

The actual instruction speed achieved in any application will depend on the speed of the computer used. The following are some representative times from a relatively slow (1.8GHz Sempron) PC.

All times are in micro-seconds per instruction.

5.2.3.1 Boolean Inputs / Edge Contacts / Compare

Instruction	Speed
STR / STRN	0.63
AND / ANDN	0.13
OR / ORN	0.13
ANDSTR / ORSTR	0.68
ANDND / ORND	0.86
ANDPD / ORPD	0.75
STRND / STRPD	1.33
ANDE / ANDNE (reg / reg)	0.13



ANDE / ANDNE (3 char string / reg)	0.21
------------------------------------	------

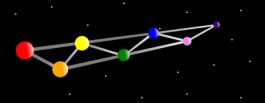
5.2.3.2 Boolean Outputs

Instruction	Single bit	2 bits	16 bits	128 bits
OUT	0.20	3.57	7.04	34.22
PD	0.88	4.38	7.95	35.55
RST / SET (rung true)	0.33	3.31	6.78	34.05
RST / SET (rung false)	0.11	0.11	0.11	0.11

5.2.3.3 Counter / Timer / Math

Instruction	Speed
CNTU	2.53
TMR	10.32
TMRA	9.38
MATHDEC ($x + y * z / a \text{ MOD } b$)	5.45
MATHDEC (SIN 0.5)	5.01
MATHHEX (LSH(reg, h2))	5.16

5.2.3.4 Shift Register



Instruction	16 bits	128 bits
SHFRG	9.13	40.94

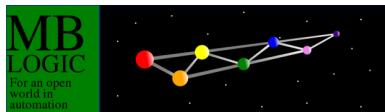
5.2.3.5 Pack / Unpack

Instruction	1 bit	16 bits
PACK	5.54	11.82
UNPACK	10.21	15.58

5.2.3.6 Single Register Copy

The single register copy operation is a multi-purpose instruction that also performs pointer (indirect addressing) and type conversion (e.g. integer to string) operations.

Instruction	Speed
COPY (constant to register without one shot) ¹	0.75
COPY (constant to register with one shot) ¹	1.5
COPY (register to register of the same type without one shot) ²	0.94
COPY (register to register of the same type with one shot) ²	1.7
COPY (register to register of different type) ³	9.9
COPY (integer to TXT conversion, 4 digits) ⁴	25



COPY (string constant to TXT registers) ⁵	24 (+ 0.4 per char)
COPY (constant to pointer)	16
COPY (register to pointer)	8.6
COPY (pointer to register)	17
COPY (pointer to pointer)	21
COPY (rung false - no operation)	0.11

¹Copying a single character constant (e.g. "A") is also a constant to register copy operation.

²Registers are considered to be of the "same" type for copy operations when a value in the source register can always be guaranteed to fit within the destination register and are of the same basic type. For example, "DS" to "DD" will always fit, but "DD" to "DS" may not fit. "DD" and "DF" are of different basic types (integer versus floating point) and so are different for copy purposes.

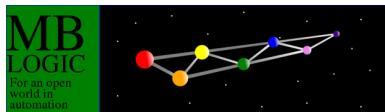
³The source type must be capable of being converted to the destination type.

⁴This involves an integer to string conversion plus a string copy.

⁵E.g. copying "ABCDE" to TXT10 - TXT14 will take $24 + (0.4 * 5) = 26$.

5.2.3.7 Multi-Register Operations

Instruction	2 regs	100 regs	1000 regs
CPYBLK	14.2	360.32	3669.33



FILL	7.97	31.74	246.88
FINDEQ	8.23	167.66	1821.93
SUM	7.88	36.05	331.68

5.2.3.8 Miscellaneous

Instruction	Speed
NETWORK (start new rung)	0.51
FOR / NEXT (per empty loop iteration)	0.08
CALL / RT (call plus return from empty subr)	3.93

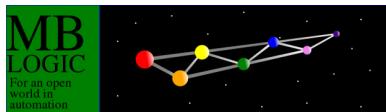
5.2.4 Instruction Set

The instruction set implemented in the "Ck" logic engine is very similar to that of the Koyo "Click". The "Ck" logic engine is not however intended to be an emulator or direct replacement for the "Click". It is intended to be a useful soft logic library that is similar enough to the "Click" that someone already familiar with one will find the other easy to learn and understand.

5.2.4.1 Instructions

The instructions are documented in detail elsewhere. The following is just a brief summary.

Instruction type	# of Instructions

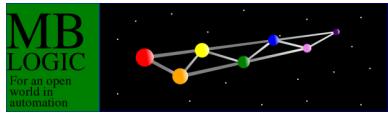


Boolean input	8
Edge contact	6
Boolean output	8
Comparison	18
Program control	8
Counter and timer	6
Copy	5
Search	12
Shift register	1
Math operators and functions	26

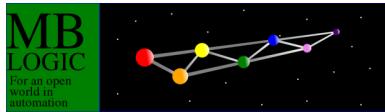
5.2.4.2 Instructions Not Implemented, or Implemented Differently

Certain instructions present in the Koyo "Click" were either not implemented in the "Ck" logic engine, or were implemented differently.

- The communications functions were not implemented. The "Ck" logic engine is part of a complete stand alone soft logic system. Communications features are provided by the rest of the supporting software.
- The SUM math function was implemented as a separate stand alone instruction rather than being integrated into the MATHDEC and MATHHEX instructions.



- Floating point math calculations are performed as double precision rather than single precision operations. This may result in more accurate, but slightly different results.
- The DRUM instructions are not implemented at this time. (They may be added in a future version).
- The watch dog timer is not implemented at this time. (This is intended to be in a future revision).
- The digital I/O system is addressed differently. The digital I/O (X, Y) is arranged as a continuous address space. The I/O registers (XD, YD) are separate address spaces from the digital I/O, and may be used for analogue and other word I/O. To address digital I/O as words, use the PACK and UNPACK instructions.
- Signed analogue I/O addresses (XS and YS) have been added. These are more generally useful than the unsigned (XD, YD) equivalents.
- There are no immediate I/O or interrupts, as these are PLC CPU on board I/O hardware features which are not present in a soft logic system.
- There is no fixed scan mode.
- Hardware related system control relay (SC) and system register (SD) addresses which relate only to "Click" hardware features are not implemented.



5.3 Addressing

5.3.1 Overview

Data for the "Ck" soft logic library is stored in a data table which is divided into distinct types. The sizes and limits on the data table and constants are defined below.

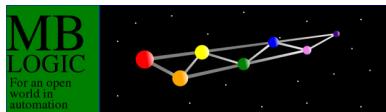
5.3.2 Data Types

Type	Size (bytes)	Min Value	Max Value
Signed integer	2	-32768	32767
Signed double integer	4	-2147483648	2147483647
Floating point	Undefined	-1.5e+308 ¹	1.5e+308 ¹
Unsigned integer (hex)	2	0	65535
Single character	1	N/A	N/A
Text string	Undefined	N/A	N/A

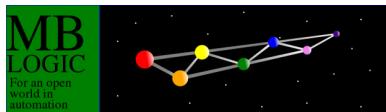
¹THe maximum constant that can be entered is +/- 1.9e307.

5.3.3 Data Table Addresses

All data is stored in data table addresses. These addresses are either "bit" (boolean) or "register" (word) addresses. Each data table begins at "1" (not "0"), and extends to the maximum address listed below.



Prefix	Description	Max	Type	Example
X	Inputs	2000	Boolean	X9
Y	Outputs	2000	Boolean	Y1023
C	Control relays	2000	Boolean	C58
T	Timer status	500	Boolean	T421
CT	Counter status	250	Boolean	CT6
SC	System control	1000	Boolean	SC21
DS	Data register	10000	Integer	DS6432
DD	Data register	2000	Double integer	DD4
DH	Data register	2000	Unsigned integer	DH100
DF	Data register	2000	Floating point	DF57
XD	Input register	125	Unsigned integer	XD1
YD	Output register	125	Unsigned integer	YD99
XS	Input register	125	Signed integer	XS1
YS	Output register	125	Signed integer	YS99



TD	Timer current value	500	Integer ¹	TD45
CTD	Counter current value	250	Double integer ¹	CTD3
SD	System data	1000	Integer	SD9
TXT	Text data	10000	ASCII	TXT7420

¹ Only positive numbers (0 to max) are allowed for these addresses.

5.3.3.1 Pointers or Indirect Addressing

Pointer (indirect) addressing is provided by the COPY instruction. See the documentation on that instruction for details.

5.3.3.2 Differences from the Koyo "Click"

The maximum address permitted by the Ck soft logic library is in some cases larger than that permitted by the Koyo "Click".

The Ck library implements the X, Y, XD, and YD addresses slightly differently from the Koyo "Click". This is because the I/O hardware is different, which means that an exact correspondence would make little sense.

The differences are as follows:

- X & Y - All addresses from 1 up to the maximum are permitted as legal addresses. The mapping of physical I/O to "X" & "Y" addresses is configuration dependent.
- XD & YD - These registers are not related to the "X" or "Y" bit addresses. That is, the states of "X" or "Y" addresses are not reflected by the values in the "XD" or "YD" registers. Instead, these registers are intended for analogue and other word I/O. The mapping of physical I/O to "XD" & "YD" addresses is configuration dependent.



- XD & YD register addresses begin at "1" in order to be consistent with the numbering system used for other registers.
- XS & YS register addresses are an additional register type. They differ from the XD and YD addresses in that they are signed rather than unsigned integers. These may be more convenient when dealing with signed integer data.
- Any data type may be used for interfacing with external I/O. For example, it is possible to transfer data directly between external I/O and C or DS addresses. X, Y, XD, YD, XS, and YS addresses however are recommended for this purpose.

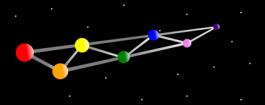
5.3.3.3 Representation

Data table addresses must be represented exactly as defined. Case is significant, so while X1 is a valid address, x1 is not. Leading zeros are not permitted. X1 is a valid address, X01 is not.

5.3.4 Constants

Many instructions which accept registers addresses as inputs will also accept constants. The type of constant must be compatible with the parameter expected by the instruction.

	Type	Example
KInt	Signed integer	125
KDInt	Signed double integer	-50000
KF	Floating point	123.456 or 1.23456E+2
KHex	Unsigned integer (hex)	f73h
KTxtChar	Single character	"A"



KTxtStr	Text string	"abc123"
---------	-------------	----------

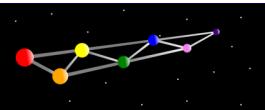
- Hexadecimal constants must have an "h" suffix to be valid. The "h" must be lower case.
 - Text constants must be enclosed in double quotes ("").
 - Integers and double integers are distinguished by the magnitude of the value. An integer constant is assumed to be a normal (single) integer unless it exceeds the maximum permitted value, in which case it is automatically promoted to a double integer. Any instruction which expects a double integer will also accept a regular integer.
 - Floating point numbers may be represented in decimal or exponential notation.
 - Leading zeros are permitted, and are not significant.
-

5.3.5 System Control Relays

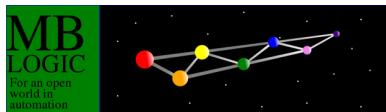
System control relays are used to provide convenience functions, or to signal error conditions.

- Error relays are set immediately by the instruction which encountered the error, and remain in effect until they are reset by another instruction which uses that relay, or until the system is restarted.
- Free running clock relays are updated at the start of each scan.
- The "always on" relay is set at the start of scan.

Address	Description
SC1	Always ON
SC2	ON for the first program scan
SC3	Turns on and off on alternate scans



SC4	Free running clock with a 10ms period
SC5	Free running clock with a 100ms period
SC6	Free running clock with a 500ms period
SC7	Free running clock with a 1sec period
SC8	Free running clock with a 1m period
SC9	Free running clock with a 1 hour period
SC19	ON if an error occurs
SC24	There is no PLC program, or the program is invalid.
SC25	The run time system is not compatible with the program version.
SC26	The watch dog timer timed out. ¹
SC27	The data table contents were lost.
SC40	An attempt was made to divide by zero.
SC43	Data overflow, underflow, or data conversion error.
SC44	An invalid address was used
SC46	A math error occurred



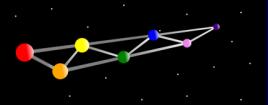
SC50	Set run mode to stop. ¹
SC51	Resets watchdog timer to zero. ¹

Notes: 1 = Not implemented at this time. The SC relays are not writable by a user instruction.

5.3.6 System Control Registers

System control registers are used to provide convenience functions, or to signal error conditions.

Address	Description
SD1	Current PLC error code ¹
SD5	Low word of runtime version ¹
SD6	High word of runtime version ¹
SD9	Counts number of scans (rolls over at 32,767)
SD10	The current scan time
SD11	The minimum scan time since starting
SD12	The maximum scan time since starting
SD20	Real time clock year



SD21	Real time clock month
SD22	Real time clock day
SD23	Real time clock day of week (where Mon = 0) ²
SD24	Real time clock hour
SD25	Real time clock minute
SD26	Real time clock second

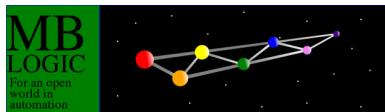
1 = Not implemented at this time.

2 = This is different than the Click hardware.

5.3.7 Subroutines

Subroutines are referenced by names which may be from 1 to 24 characters long and which may contain the characters a to z, A to Z and 1 to 9. No other characters are permitted, and no spaces are permitted within the name. Examples, "SubroutineName1", or "123ValidName". The following are invalid names "ThisNameIsMuchTooLongToBeAccepted", and "Bad&Characters*"'

There is no limit to the number of subroutines permitted, but each subroutine name must be unique.



5.4 Instructions

5.4.1 Overview

This page provides a description of each instruction list (IL) instruction used in the Ck soft logic library. "Ck" is a soft logic library which is modeled closely on the Koyo "Click" PLC. The Ck library is not intended to provide an exact emulation of the "Click". Rather it is intended to provide an instruction set which is similar enough to the "Click" that someone who is already familiar with the "Click" will understand the "Ck" library fairly easily.

The most significant difference between the two systems is the Koyo "Click" does not have a documented IL instruction set. The IL instructions used below have been interpolated from a combination of the ladder instructions used by the "Click", and the IL instructions used in the Koyo DL-205.

This document is not intended as a primer in PLC programming in general or in IL programming. The reader is assumed to be already familiar with PLC programming concepts, conventions, and terminology.

5.4.2 Instructions

Select (click on) a heading for details on any of the following instructions.

5.4.2.1 [The Logic Stack](#)

An explanation of the logic stack, and how it is used when executing boolean instructions.

[more details ...](#)

5.4.2.2 [Program Formatting Instructions](#)

- **//** - Comments.
- **Network** - Starting rungs.



[more details ...](#)

5.4.2.3 Boolean Input Instructions

- **AND** - AND bit with top of logic stack.
- **ANDN** - AND NOT bit with top of logic stack.
- **ANDSTR** - AND top two values on logic stack.
- **OR** - bit with top of logic stack.
- **ORN** - OR NOT bit with top of logic stack.
- **ORSTR** - OR top two values on logic stack.
- **STR** - Store bit onto logic stack.
- **STRN** - Store NOT bit onto logic stack.

[more details ...](#)

5.4.2.4 Edge Contact Instructions

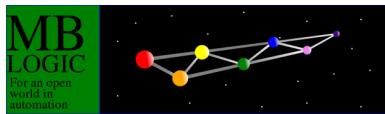
- **ANDND** - AND negative differential.
- **ANDPD** - AND positive differential.
- **ORND** - OR negative differential.
- **ORPD** - OR positive differential.
- **STRND** - STORE negative differential.
- **STRPD** - STORE positive differential.

[more details ...](#)

5.4.2.5 Boolean Output Instructions

- **OUT** - Output logic stack to bit.
- **OUT** - Output logic stack to multiple bits.
- **PD** - Output logic stack one shot.
- **PD** - Output logic stack one shot to multiple bits.
- **RST** - Reset bit if logic stack true.
- **RST** - Reset multiple bits if logic stack true.
- **SET** - Set bit if logic stack true.
- **SET** - Set multiple bits if logic stack true.

[more details ...](#)



5.4.2.6 Comparison Instructions

- **ANDE** - AND if parm1 equals param2.
- **ANDGE** - AND if parm1 >= param2.
- **ANDGT** - AND if parm1 > param2.
- **ANDLE** - AND if parm1 <= param2.
- **ANDLT** - AND if parm1 < param2.
- **ANDNE** - AND if parm1 is not equal to param2.
- **ORE** - OR if parm1 equals param2.
- **ORGE** - OR if parm1 >= param2.
- **ORGТ** - OR if parm1 > param2.
- **ORLE** - OR if parm1 <= param2.
- **ORLT** - OR if parm1 < param2.
- **ORNE** - OR if parm1 is not equal to param2.
- **STRE** - STR if parm1 equals param2.
- **STRGE** - STR if parm1 >= param2.
- **STRGT** - STR if parm1 > param2.
- **STRLE** - STR if parm1 <= param2.
- **STRLT** - STR if parm1 < param2.
- **STRNE** - STR if parm1 is not equal to param2.

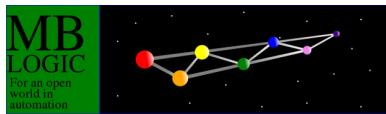
[more details ...](#)

5.4.2.7 Program Control Instructions

- **END** - Program end .
- **ENDC** - Program end conditional.
- **FOR** - For/next loop.
- **NEXT** - Next in For/next loop.
- **SBR** - Define a subroutine.
- **CALL** - Call subroutine.
- **RT** - Return from subroutine.
- **RTC** - Return from subroutine conditional.

[more details ...](#)

5.4.2.8 Counter and Timer Instructions



- **CNTU** - Count up.
- **CNTD** - Count down.
- **UDC** - Up/down counter.
- **TMR** - On delay timer.
- **TMRA** - On delay accumulating timer.
- **TMROFF** - Off delay timer.

[more details ...](#)

5.4.2.9 *Copy Instructions*

- **COPY** - Copy a single value to a register.
- **CPYBLK** - Copy a block of data.
- **FILL** - Fill a block of data.
- **PACK** - Pack bits into a register.
- **UNPACK** - Unpack bits from a register.

[more details ...](#)

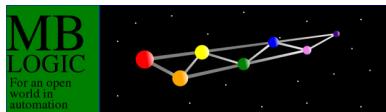
5.4.2.10 *Search Instructions*

- **FINDEQ** - Search table for equal to.
- **FINDGE** - Search table for \geq .
- **FINDGT** - Search table for $>$.
- **FINDLE** - Search table for \leq .
- **FINDLT** - Search table for $<$.
- **FINDNE** - Search table for not equal.
- **FINDIEQ** - Incremental search table for equal to.
- **FINDIGE** - Incremental search table for \geq .
- **FINDIGT** - Incremental search table for $>$.
- **FINDILE** - Incremental search table for \leq .
- **FINDILT** - Incremental search table for $<$.
- **FINDINE** - Incremental search table for not equal.

[more details ...](#)

5.4.2.11 *Shift Register Instructions*

- **SHFRG** - Shift register.



[more details ...](#)

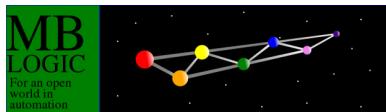
5.4.2.12 [*Math Instructions*](#)

- **MATHDEC** - Decimal math.
- **MATHHEX** - Hexadecimal math.
- **SUM** - Sum a range of registers.

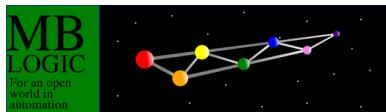
[more details ...](#)

5.4.3 Alphabetic Instruction Summary

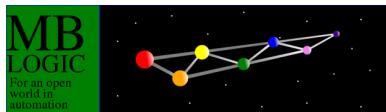
Instruction	Instr Type	Description
//	Program Formatting	Comment line
AND	Boolean input	AND bit with top of logic stack
ANDE	Compare	AND if parm1 equals param2
ANDGE	Compare	AND if parm1 >= param2
ANDGT	Compare	AND if parm1 > param2
ANDLE	Compare	AND if parm1 <= param2
ANDLT	Compare	AND if parm1 < param2
ANDN	Boolean input	AND NOT bit with top of logic stack
ANDND	Edge contact	AND negative differential



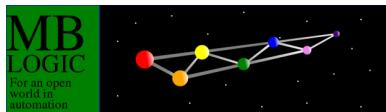
ANDNE	Compare	AND if parm1 is not equal to parm2
ANDPD	Edge contact	AND positive differential
ANDSTR	Boolean input	AND top two values on logic stack
CALL	Program control	Call subroutine
CNTD	Counter/Timer	Count down
CNTU	Counter/Timer	Count up
COPY	Copy	Copy a single value to a register
CPYBLK	Copy	Copy a block of data
END	Program control	Program end
ENDC	Program control	Program end conditional
FILL	Copy	Fill a block of data
FINDEQ	Search	Search table for equal to
FINDGE	Search	Search table for >=
FINDGT	Search	Search table for >
FINDIEQ	Search	Incremental search table for equal to



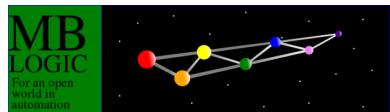
FINDIGE	Search	Incremental search table for >=
FINDIGT	Search	Incremental search table for >
FINDILE	Search	Incremental search table for <=
FINDILT	Search	Incremental search table for <
FINDINE	Search	Incremental search table for not equal
FINDLE	Search	Search table for <=
FINDLT	Search	Search table for <
FINDNE	Search	Search table for not equal
FOR	Program control	For/next loop
MATHDEC	Math	Decimal math
MATHHEX	Math	Hexadecimal math
NETWORK	Program Formatting	Network
NEXT	Program control	Next in For/next loop
OR	Boolean input	OR bit with top of logic stack
ORE	Compare	OR if parm1 equals parm2



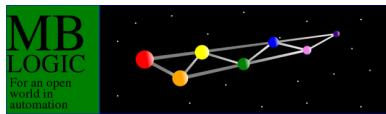
ORGE	Compare	OR if parm1 >= parm2
ORGТ	Compare	OR if parm1 > parm2
ORLE	Compare	OR if parm1 <= parm2
ORLT	Compare	OR if parm1 < parm2
ORN	Boolean input	OR NOT bit with top of logic stack
ORND	Edge contact	OR negative differential
ORNE	Compare	OR if parm1 is not equal to parm2
ORPD	Edge contact	OR positive differential
ORSTR	Boolean input	OR top two values on logic stack
OUT	Boolean output	Output logic stack to bit
OUT	Boolean output	Output logic stack to multiple bits
PACK	Copy	Pack bits into a register
PD	Boolean output	Output logic stack one shot to multiple bits
PD	Boolean output	Output logic stack one shot
RST	Boolean output	Reset multiple bits if logic stack true



RST	Boolean output	Reset bit if logic stack true
RT	Program control	Return from subroutine
RTC	Program control	Return from subroutine conditional
SBR	Program control	Define a subroutine
SET	Boolean output	Set multiple bits if logic stack true
SET	Boolean output	Set bit if logic stack true
SHFRG	Shift register	Shift register move bits to right
STR	Boolean input	Store bit onto logic stack
STRE	Compare	STR if parm1 equals parm2
STRGE	Compare	STR if parm1 >= parm2
STRGT	Compare	STR if parm1 > parm2
STRLE	Compare	STR if parm1 <= parm2
STRLT	Compare	STR if parm1 < parm2
STRN	Boolean input	Store NOT bit onto logic stack
STRND	Edge contact	STORE negative differential



STRNE	Compare	STR if parm1 is not equal to parm2
STRPD	Edge contact	STORE positive differential
SUM	Math	Sum a range of registers
TMR	Counter/Timer	On delay timer
TMRA	Counter/Timer	On delay accumulating timer
TMROFF	Counter/Timer	Off delay timer
UDC	Counter/Timer	Up/down counter
UNPACK	Copy	Unpack bits from a register



5.5 Programming

5.5.1 Overview

The purpose of a soft logic system is to run programs written in programming languages used by PLCs. The following describes the format and structure of these programs.

5.5.2 PLC Programming Languages

PLC programs are typically displayed in either ladder (LAD) or instruction list (IL). However what, a PLC or soft logic system actually executes is actually IL (or something close to it). Ladder is simply a graphical means of displaying IL.

5.5.3 Source File Format

The file format used is plain ASCII text files. These may be created using the built in IL editing facilities in the status system, or with any text editor capable of creating plain text files.

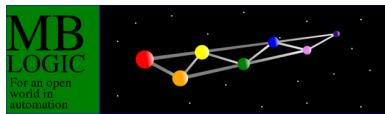
5.5.3.1 File Format

Program source files must be in plain ASCII text with one instruction per line. Each line must be separated by a newline character.

5.5.3.2 Intermediate Formats

Although the soft logic system generates intermediate forms of the program before executing, these are not stored permanently. Only the plain source format is kept and used each time the system is started or the program reloaded.

5.5.3.3 File Names



The name of the file used to store the soft logic program source code is defined in the soft logic configuration system. See the documentation on that feature for more details on this subject.

5.5.3.4 Number of Source Files

A complete program must be in a single file, and this file must contain all subroutines referenced by the program.

5.5.4 Program Symbols

The following defines some general rules which must be followed when writing soft logic programs.

5.5.4.1 Comments

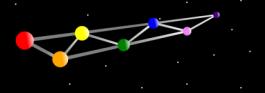
Comments are defined by the comment symbol "//". The comment symbol must be followed by a space. Comments must start at the beginning of a line. Anything following a comment symbol on the same line is ignored.

```
// This is a comment.  
// And this is another comment.
```

5.5.4.2 Rungs or Networks

PLC programs are normally separated into "rungs" or (in IEC terminology) "networks". In this soft logic system, a rung symbol is an instruction and executes code which prepares the system to execute the IL code contained in the rung and also stores diagnostic information.

Rungs are normally numbered, although the rung numbers used can be arbitrary and in any order. They may even be duplicated. However, since the system stores the rung numbers to provide diagnostic information in the event of an error, it is normally advantageous to use unique, ordered rung numbers in a program. Empty rungs are permitted.



```

NETWORK 14
// This is a rung.
STR X1
AND C2
OUT Y33

NETWORK 15
// This rung is empty.

NETWORK 16
// And this is another rung.
STR X14
OUT Y12

```

5.5.4.3 Program Instructions

Anything which is not a comment is treated as an instruction (including the start of rungs or networks). Instructions must follow these rules:

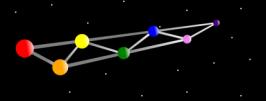
- Each instruction must start at the beginning of a line.
- There must be no more than one instruction per line.
- Parameters are separated from instructions by spaces.
- Everything on the same line following an instruction is considered to be a parameter for that instruction.
- An instruction may be of any length, but must not contain spaces.

Each instruction must either have a unique symbol or be able to be uniquely identified by its parameter types. It must not rely on the context of other adjacent instructions to identify it. When an instruction is capable of accepting multiple incompatible parameter types, each variant is considered to be a different instruction. When loading a soft logic program, the soft logic system will search its list of valid instructions for the first one which matches both the instruction symbol and the parameter types. If no match is found, it is considered to be an unknown instruction.

```

// This is just an example and may not match any particular PLC.
// This is an instruction using a boolean variable.
AND X1
// This is a different instruction using a word variable.
AND V10000
// This is a different instruction using a pointer variable.
AND P10000
// This is a different instruction using a constant.
AND K1234

```



```
// This is an invalid instruction as the parameter type does not match.  
AND R10.5
```

5.5.4.4 Data Table Addresses

The soft logic system treats all data table addresses as "labels". An address label is considered to be an arbitrary name with no significance other than as a unique identifier. This means that the soft logic system does not attempt to infer relationships between addresses based on any numbers contained in the address label. This allows the system to deal with different address formats without having to "understand" them.

This also means however that addresses which may have the same "numerical" value but which are written differently are treated as different addresses. For example, "X1" and "X01" are considered to be different addresses. Therefore, programs must be written to a single form of each address.

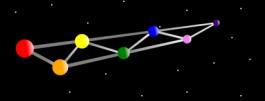
```
// This instruction is using a valid address.  
STR X1  
// The address used with this instruction is not recognised.  
STR X01
```

5.5.4.5 Subroutines

When the system starts to read in a soft logic program source file, the first instructions encountered are considered to be part of the "main" routine. If no subroutine instruction is encountered, then the entire program is considered to be a single "main" routine.

The start of a subroutine is indicated by the "instruction" which defines the start of a subroutine. All code following a subroutine instruction is considered to be part of that subroutine until either a new subroutine instruction is encountered, or until the end of the source file.

This means that all subroutines must be at the end of a file, and that subroutines may not be nested (you may not define a subroutine inside another subroutine).



```
// This is the start of the main routine.  
NETWORK 1  
STR X1  
OUT Y1  
  
NETWORK 2  
// This calls a subroutine.  
STR X31  
CALL Sub1  
  
NETWORK 3  
// The program scan ends here.  
END  
  
// #####  
SBR Sub1  
// This is a subroutine.  
NETWORK 1  
STR Y1  
AND Y2  
OUT C16  
RT  
  
// This is the end of the program.
```

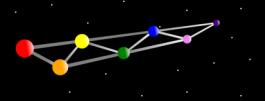
5.5.5 Ladder Representation

Soft logic programs can be displayed in either IL (instruction list) or ladder format (see the "Status System" for more details on this). The ladder display system will automatically convert IL logic into ladder format if it is possible to do so. However, since IL is more flexible than ladder, it is possible to write valid IL logic in a manner which cannot be converted into ladder (this is a problem common to all PLCs). When a rung of logic cannot be displayed as ladder, the system will default to displaying it in IL.

5.5.5.1 Rules for Ladder Versus IL Display

The following rules govern whether an IL program may be displayed in ladder format:

1. Output instructions may not be followed by input instructions in the same rung. A rung may contain more than one output instruction, but all inputs must precede any outputs.



2. Any output instruction which takes more than one logic input (e.g. shift register, counters, etc.) must be the only output in the rung.
3. There must be no more than 8 inputs in series. More than this will not fit on the page.
4. Errors in the soft logic program created by the user may result in logic cannot be displayed in ladder (as well as not executing correctly).

When a rung cannot be displayed in ladder format, it will instead be displayed in IL. Each rung is dealt with individually, and problems with one rung will not affect subsequent rungs.

5.5.5.2 Ladder Versus IL Instructions

Each individual ladder instruction is equivalent to a single IL instruction. Ladder is simply an alternative way of representing IL. See the documentation on instructions for more details.

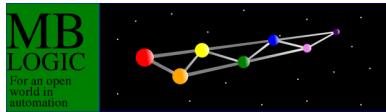
5.5.5.3 Comments

In IL format, comments may be inserted anywhere in a program. However, when the rung is displayed in ladder format, all comments are gathered together in a block immediately below the start of the rung. This does not affect the file in which the program is stored. It only affects how the comments are displayed. All comments between the start of a rung (a NETWORK instruction) and the start of the next rung (or subroutine) belong to that rung.

Any comments which exist between a subroutine definition (SBR instruction) and the first rung in the subroutine are considered to belong to the subroutine definition and are displayed there. Any comments which exist between the start of the program and the first rung are considered to belong to the main program, and are treated similarly.

5.5.5.4 Unconditional Outputs

Some instructions (e.g. RT - unconditional return from subroutine) are "unconditional". That is, they execute regardless of the current logic state. If these instructions are inserted into a rung together with input instructions and conditional outputs, they will attach themselves to the conditional outputs. **However**, this does



not affect how the unconditional instructions work. They will continue to execute regardless of the logic state of the rest of the rung.

5.5.5.5 Name of Program

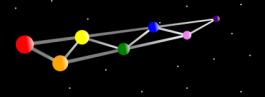
The file name of the program is automatically inserted into the heading of the main program.

5.5.6 Example Program

The following shows a simple program.

Example:

```
// Demo program for soft logic.  
NETWORK 1  
// This is a counter.  
STR X1  
STR SC1  
CNTU CT100 50  
  
NETWORK 2  
// This is a timer  
STR SC1  
TMR T5 329 ms  
  
STR T5  
OUT Y367  
  
NETWORK 3  
// This calls a subroutine.  
STR T5  
CALL Sub1  
  
NETWORK 4  
// This just executes some ladder logic.  
STR SC1  
ORN C2  
AND X17  
OUT Y1  
  
NETWORK 5  
END  
  
SBR Sub1
```

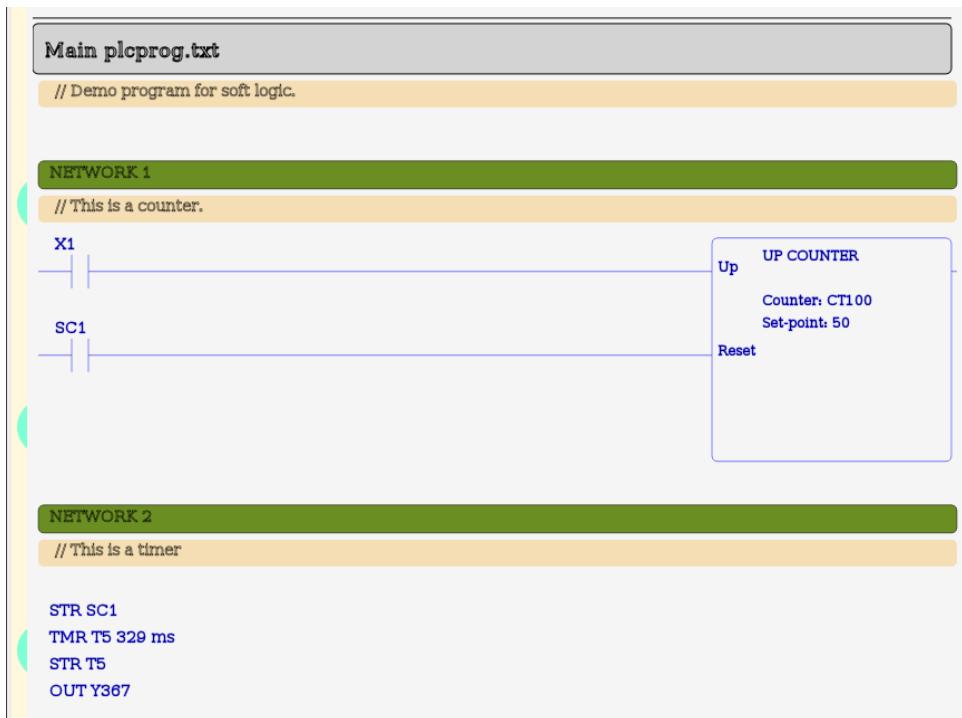


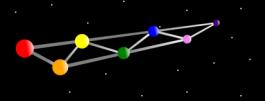
```
// This is a subroutine.
NETWORK 1
STR SC1
MATHDEC DS100 0 SQRT(DD1 + 25)

NETWORK 2
STRGE DS100 1
OUT C50

NETWORK 3
RT
```

This shows the same program as ladder. Note that the program is appearing as a mixture of ladder and IL, as parts of the program were written in a manner with no ladder equivalent.





NETWORK 3

// This calls a subroutine.

T5

CALL
Sub1

NETWORK 4

// This just executes some ladder logic.

SC1



Y1
OUT

NETWORK 5

END

SBR Sub1

// This is a subroutine.

NETWORK 1

SC1

MATHDEC 0
DS100 =
SQRT(DD1 + 25)

NETWORK 2

DS100

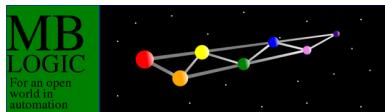
1

C50

OUT

NETWORK 3

RETURN RT



5.6 Configuration

5.6.1 Overview:

The soft logic system has its own data table which is independent of the main system (or communications) data table). The soft logic IO configuration provides the connection between the two data tables. Without a soft logic IO configuration, the soft logic program will run, but it will have no way to connect to real world IO or to the HMI.

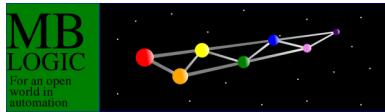
5.6.2 Configuration

Configuration is performed through a configuration file called "**"mblogic.config"**". This file holds all the data table names, addresses, and other required information. This is a plain text file which can be edited with any text editor. The configuration file is defined as follows:

- One line per parameter. Each parameter must appear on a separate line.
 - Comments. Any line beginning with a "hash" ("#") character is a comment and is ignored.
 - Sections. Sections begin a series of related configuration values. A section name is an identifier which is enclosed by square brackets ("[]"). A section includes all items until the beginning of a new section, or until the end of the file.
 - Configuration items. Any line following the beginning of a section is considered to be an item which belongs to that section (excluding comments). Configuration items are key/value pairs in the format "key=value".
-

5.6.3 System Identification

The configuration must include a section named "&system". The system identification section is used to define parameters which apply to the overall soft logic system. These factors are:



- type - This defines which soft logic library will be used to interpret the soft logic program. Only the "ck" library exists at present. Others may be added in future.
- plcprog - This determines the name used for the softlogic program file. Set this to the name you wish to use for your program file.
- scan - This is the **target** scan rate in milli-seconds. The system will attempt to run your soft logic program once every 'x' milli-seconds, where 'x' is the value you provide. See the comments on application considerations below with regards to how to set this value.

Example:

```
[&system]
type=ck
plcprog=plcprog.txt
scan=50
```

5.6.4 Data Table Save/Restore

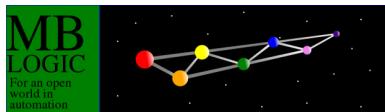
The configuration must include a section named "&logicsave". This section is used to define parameters which apply to the data table save and restore system (see the help topic on "Soft Logic Data Table Save" for details on how this works). These factors are:

- updateinterval - This defines the minimum update interval. This must be a valid integer or floating point number.
- wordaddr - This is a list of soft logic word addresses which are to be saved to disk. These may be any valid word (not boolean) addresses. Individual addresses must be separated by commas.

Example:

```
[&logicsave]
updateinterval=2.0
wordaddr=DS1,DS2,DS3,DS4,DS5,DS6,DS7,DS8,DS9,DS10
```

5.6.5 IO Sections

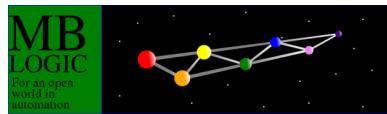


The remainder of the configuration consists of "IO sections". An IO section defines a group of addresses which are two be transferred between the two data tables. The section names are arbitrary, and the user is free to select any name desired provided it does not start with an ampersand ("&"). Names starting with an ampersand (e.g. "&system") are considered special and reserved for use by the soft logic system.

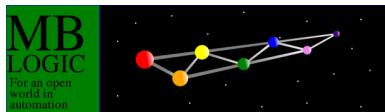
5.6.5.1 Address Types

Address types determine how data is stored in the system data table. This includes the number of storage locations (e.g. registers) and the byte ordering. For numeric address types which occupy multiple registers, the least significant word is stored in the platform's native format. For Intel type CPUs (big endian), the first (lower address) register. For string data types, characters are stored in consecutively increasing addresses. The following address types are currently defined:

Address Type	Data Type	Storage Location	# Storage Elements	Soft Logic Address Compatibility	Extended Data Type
coil	boolean	coil	1	X, Y, C	No
discrete	boolean	discrete inputs	1	X, Y, C	No
holdingreg	signed 16 bit integer	holding register	1	'DS', 'XD', 'YD', 'XS', 'YS'	No
inputreg	signed 16 bit integer	input register	1	'DS', 'XD', 'YD', 'XS', 'YS'	No
holdingreg32	signed 32 bit integer	holding register	2	DD	Yes



inputreg32	signed 32 bit integer	input register	2	DD	Yes
holdingregfloat	single precision (32 bit) floating point	holding register	2	DF	Yes
inputregfloat	single precision (32 bit) floating point	input register	2	DF	Yes
holdingregdouble	double precision (64 bit) floating point	holding register	4	DF	Yes
inputregdouble	double precision (64 bit) floating point	input register	4	DF	Yes
holdingregstr8	string	holding register	2 characters per register	TXT	Yes

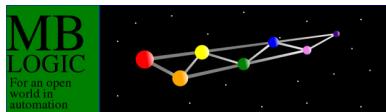


inputregstr8	string	input register	2 characters per register	TXT	Yes
holdingregstr16	string	holding register	1 character per register	TXT	Yes
inputregstr16	string	input register	1 character per register	TXT	Yes

5.6.5.2 Configuring IO Sections

An IO section must contain the following items:

- addrtype - This defines the type of system data table address. It must be one of the address types listed above.
- base - This is the beginning of the block of system data table addresses defined. The number of addresses used will depend on the number of soft logic addresses defined in the "logitable" parameter.
- action - This must be either "read" or "write". Specifying "read" will cause the IO section to transfer data **from** the system data table **to** the soft logic table. Specifying "write" will cause the IO section to transfer data **to** the system data table **from** the soft logic table.
- logitable - This must be one or more valid soft logic addresses, with multiple addresses separated by commas (e.g. "X1,Y10,C9". This defines the soft logic data table addresses that data will be transferred to or from. Addresses do not need to be in any particular order, but they must all be of a compatible type. Native data types may define multiple addresses. Extended data types are limited to a single address.
- strlen - This parameter is only used with string length. It defines the maximum number of characters for the string. This parameter is ignored for other data types.



Example:

```
[PBs]
addrtype=coil
base=32100
action=read
logictable=X1,X2,X3,X4,X5,X6,X7,X8

[PartCount]
addrtype=holdingreg32
base=42
action=write
logictable=DS101

[PartName]
addrtype=inputregstr16
base=1234
action=read
logictable=TXT99
strlen=10
```

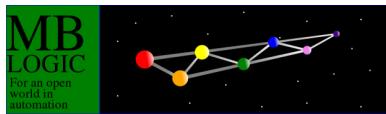
Review the documentation on the system data table before deciding which registers, coils, or discrete inputs to use.

5.6.6 Application Considerations

5.6.6.1 Grouping IO Addresses

A configuration must include one (and only one) "system identification" ("&system") section. It may include as many "IO" sections as desired. The IO sections may be given meaningful names (e.g. "PushButtons"). However, the primary purpose of a section is to group associated addresses together so they can be dealt with efficiently. A good approach is to have each IO section represent an field device module or IO card(s). This of course assumes that the client and server commands for the system data table have been grouped in this manner.

Addresses in the system data table are handled most efficiently when they are read or written in consecutive blocks.



For the soft logic data table, it does not matter what order addresses are in, or whether they are grouped together. This means that if you wish to group or organise addresses in your program differently in your soft logic program than they are organised in the actual field devices, it is best to do this in the soft logic configuration and to leave the actual IO communications in their natural order.

5.6.6.2 Soft Logic Program Name

The name used for the soft logic program is defined in the system parameters (see above). Any name can be used, provided it doesn't conflict with the name of a file which is part of the system (e.g. one of the configuration files). The file can also have any file extension desired or even no file extension. This allows names such as "Machine_021.prog" or "Water_System.txt" (or whatever else is desired).

5.6.6.3 Program Scan Rate

The soft logic program target scan rate is defined in the system parameters (see above). The target scan rate means that the system will attempt to run one scan of the soft logic program at the requested interval. Note that this is the **desired** scan rate, not the measured rate.

A soft logic system differs from a conventional PLC in that the soft logic system is running on a general purpose computer platform. There may be other things such as HMIs, databases, and other tasks running on the same platform. On a conventional PLC, to add more features you would normally just add more hardware modules (with each feature needing its own hardware to run on). A soft logic system will often have to share the hardware with other applications. This means that the soft logic system cannot assume that all the computing resources are available for its exclusive use.

For a soft logic system, this means that taking up 100% of the CPU capacity to scan the program as fast as possible is not a good idea. In a new application, the target scan rate should be initially set to a larger value (e.g. 50 msec) and then adjusted downwards to smaller values as needed. The amount of CPU capacity being used can be measured by using the normal operating system utilities for whatever operating system you are using (e.g. "top" or "System Monitor" for Linux, or the equivalent utility for MS Windows).



The program scan rate is set at system start up and cannot be changed while the system is running. This means that to change the scan rate, you need to stop and restart the system.

5.6.6.4 *Changing the Configuration*

The soft logic IO configuration is specified in a configuration file. This file is read when the system starts up. However, the configuration can also be changed while the system is running by reloading the configuration using the system "status" interface. Details of how to do this may be found in the help pages which are part of the status system.

5.6.7 Complete Example

The following shows a complete example.

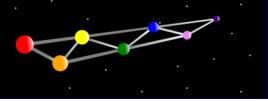
```
# Test configuration for soft logic system.
# 17-Jun-2009

# System parameters.
# Set the type of soft logic, the default PLC program name,
# and the target scan rate.
[&system]
type=ck
plcprog=plcprog.txt
scan=50

# Data table values to save to disk, along with
# the minimum update interval.
[&logicsave]
updateinterval=2.0
wordaddr=DD1,DF2,DS10

#####
# Each section must be given a unique name. The name can be
descriptive, but
# must not start with an ampersand ('&').

# Get the push buttons.
[PBs]
addrtype=coil
base=32100
action=read
logictable=X1,X2,X3,X4,X5,X6,X7,X8
```



```

# Set the pilot lights.
[PLs]
addrtype=discrete
base=32100
action=write
logictable=Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8

# Get PB4 register.
[PB4]
addrtype=holdingreg
base=32200
action=read
logictable=XD1

# Set PL4 register.
[PL4]
addrtype=inputreg
base=32200
action=write
logictable=YD1

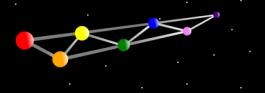
# Tank Readings. These are Tank1 level, Tank2 level, and Pump Speed
(actual).
[Tanks]
addrtype=inputreg
base=32210
action=write
logictable=YS10,YS11,YS12

# Pump command speed.
[PumpCommand]
addrtype=holdingreg
base=32210
action=read
logictable=XS10

#####
# Strip charts.
[StripCharts]
addrtype=inputreg
base=32213
action=write
logictable=YS20,YS21

#####
# Events.
[Events]
addrtype=coil
base=32300
action=write
logictable=Y20,Y21,Y22,Y23,Y24,Y25,Y26,Y27,Y28

```



```

# Alarms.
[Alarms]
addrtype=coil
base=32400
action=write
logictable=Y30,Y31,Y32,Y33,Y34,Y35,Y36,Y37,Y38
#####
# Test the extended data types.

# 32 bit integer.
[HRIInt32Read]
addrtype=holdingreg32
base=32215
action=read
logictable=DD10

[IRIInt32Read]
addrtype=inputreg32
base=32215
action=read
logictable=DD11

# Single precision floating point.
[HRFloat32Read]
addrtype=holdingregfloat
base=32217
action=read
logictable=DF10

[IRFloat32Read]
addrtype=inputregfloat
base=32217
action=read
logictable=DF11

# Double precision floating point.
[HRFloat64Read]
addrtype=holdingregdouble
base=32219
action=read
logictable=DF13

[IRFloat64Read]
addrtype=inputregdouble
base=32219
action=read
logictable=DF14

# Strings
[HRStrin8Read]
addrtype=holdinggregstr8

```



```
base=32223
action=read
logictable=TXT10
strlen=10

[HRStrin8Write]
addrtype=inputregstr8
base=32223
action=write
logictable=TXT50
strlen=10

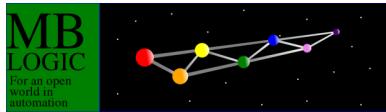
[Testholdingregstr16]
addrtype=holdingregstr16
base=32233
action=read
logictable=TXT100
strlen=14

[Testinputregstr16]
addrtype=inputregstr16
base=32233
action=write
logictable=TXT150
strlen=14

[Testholdingregstr16]
addrtype=holdingregstr16
base=32233
action=read
logictable=TXT100
strlen=14

[Testinputregstr16]
addrtype=inputregstr16
base=32233
action=write
logictable=TXT150
strlen=14

#####
#####
```



5.7 Program Monitoring & Reloading

5.7.1 Overview

The soft logic IO configuration and the soft logic program can be monitored and modified by using the "Status" interface. The status interface allows the user to control and monitor the system through a web interface. The status interface has its own help pages which describe how to operate it. This page will just review the general capabilities.

5.7.2 Programming and Trouble Shooting

5.7.2.1 *Writing Programs*

Soft logic programs are stored as instruction list (IL) plain text files. These files may be created or edited using the on-line editor built into the status system. See the help sections on "System Status" for more information.

Alternatively, since the files are plain text they may also be edited with any editor capable of using plain text files.

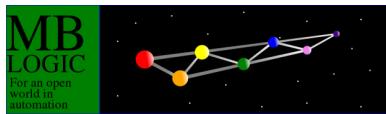
5.7.2.2 *Viewing Programs and Cross References*

The status system also provides a web based interface for viewing programs in IL and ladder formats. It also provides program cross references.

5.7.3 Starting the Soft Logic Program

The soft logic program is started automatically when the system starts up. The file name is set in the soft logic IO configuration file (see the help page on that topic for more details).

5.7.4 Reloading the Program



The soft logic program can be restarted and reloaded while the system is running by using the "Status" web interface. This does not reinitialise or change the values in the system data table or the soft logic data table. This means that reloading the soft logic program (e.g. after making a change to it) is effectively a form of "on-line programming", because it permits a program to be changed while the system is running.

It is important to remember though, that making a change to a running system may cause unexpected side effects. Be sure to exercise caution when using this feature.

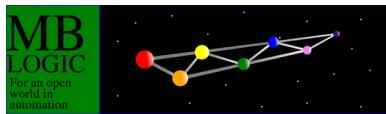
"One shot" (differentiation) instructions may lose their previous state and re-initialise. This is because with the "Ck" logic library (as with the style of conventional PLCs on which it is modelled), the state of the instruction is held with the instructions themselves, and not in the normal data table. When the program is reloaded, the previous state of these instructions is lost.

The following items are changed when the program is reloaded:

- The "first scan" flag is set to "true" ("1").
- The scan counter is reset to zero.
- The scan timers are reset to their default values.

5.7.5 Viewing Data Table Contents

The "Status" interface provides a means of viewing the contents of any data table address. See the status system documentation for more information.



5.8 Data Table Save & Reload

5.8.1 Overview

A soft logic system on a PC runs in volatile memory. The soft logic data table save/restore feature saves selected data table addresses to be saved to disk and restored when the system re-starts. Data table save and restore is part of the soft logic I/O configuration and does not require the use of any instructions in the soft logic program itself.

5.8.2 Data Table Save Configuration

The soft logic data save is configured in the soft logic I/O configuration. There are two different save parameters. One parameter is the list of addresses which are to be saved, and the second is the minimum interval between updates. See the help page on soft logic configuration for details on configuration syntax.

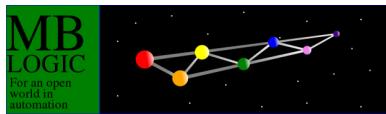
5.8.2.1 Addresses

The addresses to save are specified individually. Any of the following word address may be saved:

- DS
- DH
- DD
- DF
- TXT

Boolean addresses may not be saved. Some types of word addresses may not be saved because they are normally used for I/O interface (e.g. XD, YD), they contain system data (e.g. SD), or the complete internal state of the address cannot be guaranteed to be restored properly by simply restoring the data value (e.g. TD, CTD).

Each address is specified individually. There is no limit to the number of addresses which may be saved. However, saving an excessive number of addresses may cause



slow down the system to an unacceptable degree. The system does not check to see if the data table addresses are actually used in the soft logic program.

5.8.2.2 Update Interval

The update interval is also specified as a parameter in the I/O configuration. The update interval is the *minimum* time period between saving changes. This can be used to limit the rate at which updates are saved to disk.

The update interval is specified in seconds. Fractions of seconds may be expressed as decimal values. The *minimum* permitted update interval is 1.0 seconds. Values between 0 and 1 will be changed to 1 second. A negative value will disable data table save operations.

5.8.3 Save and Restore Process

5.8.3.1 System Start Up

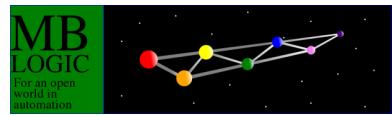
When the system starts up, the soft logic configuration is read for the list of data table addresses to be monitored and the update interval. Then, the values previously saved are restored to the soft logic data table prior to the soft logic system being started. Any addresses which are no longer monitored are purged from the database. Unused addresses are also purged whenever a change in the configuration takes place.

5.8.3.2 Saving Data Table Values

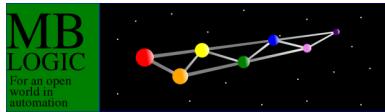
At the end of each soft logic scan the monitored values are written to disk provided (1) data table save is enabled (the update interval is greater than 0), and (2) the minimum update interval since the last save has expired, and (3) one or more data values have changed since the last update. Only changed values are written to disk.

5.8.3.3 Data Store File

Monitored data table values are stored in a file called "**mblogic.dtable**". The file name is fixed and cannot be changed. This file will be located in the same directory as the



configuration files and soft logic program. The file will be automatically created if it is not already present. The file is in a binary database format and should not be altered.



6 HMI

6.1 Overview

6.1.1 Server Configuration

This describes how to configure the server to work with an HMI application. Also see the "System Status" documentation for information on how to use the web interface to view and edit configurations.

- [HMI Server Configuration](#)
-

6.1.2 Alarm and Event History

Alarm and event history can be queried from the database using the provided web based history query system.

- [Alarm and Event History](#)
-

6.1.3 Creating HMI Web Pages

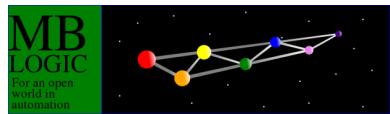
The graphical portion of the HMI can be created using the [Inkscape](#) vector graphics editor together with the provided graphical widgets (push buttons, pilot lights, etc.). In addition you can create your own custom graphics and widgets.

The completed drawing can be assembled into a finished HMI web page using MBLogic HMIBuilder. HMIBuilder is a separate program which runs as a traditional GUI program on your PC.

6.1.3.1 MBLogic HMIBuilder

- [MBLogic HMIBuilder](#) - Assembling HMI web pages using MBLogic HMIBuilder.

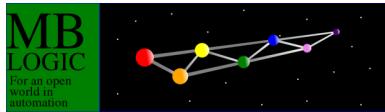
6.1.3.2 HMI Javascript Client Libraries



- [HMI Javascript Client Libraries](#) - The HMI Javascript client libraries.

6.1.3.3 Other Resources

- [Basic Concepts](#) - This section describes the basic concepts behind the operation of a web based HMI. This section is of interest to those interested in understanding the underlying principles of operation.



6.2 HMI Configuration

6.2.1 Overview

The HMI server is configured by means of a text file. The format and features of that configuration file are described here.

6.2.2 Configuration

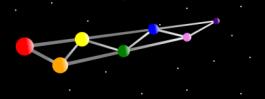
Server configuration is performed through a configuration file. This file holds all the tag names, addresses, and other definitions used by the server. Any tag names or addresses which are not contained in the configuration file are not recognised by the server. The configuration file is defined as follows:

- One line per parameter. Each parameter must appear on a separate line.
 - Comments. Any line beginning with a "hash" ("#") character is a comment and is ignored.
 - Sections. Sections begin a series of related configuration values. A section name is an identifier which is enclosed by square brackets ("[]"). A section includes all items until the beginning of a new section, or until the end of the file.
 - Configuration items. Any line following the beginning of a section is considered to be an item which belongs to that section (excluding comments). Configuration items are key/value pairs in the format "key=value".
-

6.2.3 Configuring Reserved Tags

Certain tags are defined as "reserved tags". These include tags for defining the client version and the server id. The configuration system allows "clientversion" and "serverid" to be defined by the server. For "clientversion" there is one item "ver" which defines the version. For "serverid", there is also one item "id" which allows the server id to be defined.

```
# Client page(s) version.
```



```
[clientversion]
ver=Ver 0.1 Demo

# Server ID.
[serverid]
id=demo test server
```

6.2.4 Configuring Address Tags

Any section name other than a reserved tag or events, alarms, or erplist is considered to be an address tag. Each data table address which is to be accessed via the HMI protocol must have an address tag.

Each address tag has a series of items defining the data table type, data table address, and various other characteristics. The number and type of items required will vary depending on the data type.

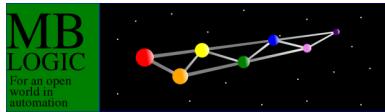
6.2.4.1 Data Types

Data types determine the format for data which transmitted between the server and the client. The system will convert between integer and floating point if necessary. It does **not** determine server data table storage size or packing. The following data types are recognised for Modbus communications:

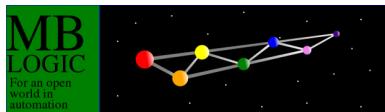
- boolean - This is for bit (0/1 or true/false) values.
- integer - Integer data types.
- float - Floating point data types.
- string - String data types.

6.2.4.2 Address Types

Address types determine how data is stored in the server data table. This includes the number of storage locations (e.g. registers) and the byte ordering. For numeric address types which occupy multiple registers, the least significant word is stored in the platform's native format. For Intel type CPUs (big endian), the first (lower address) register. For string data types, characters are stored in consecutively increasing addresses. The following data types are recognised for Modbus communications:



Address Type	Data Type	Storage Location	# Storage Elements	Extended Data Type
coil	boolean	coil	1	No
discrete	boolean	discrete inputs	1	No
holdingreg	signed 16 bit integer	holding register	1	No
inputreg	signed 16 bit integer	input register	1	No
holdingreg32	signed 32 bit integer	holding register	2	Yes
inputreg32	signed 32 bit integer	input register	2	Yes
holdingregfloat	single precision (32 bit) floating point	holding register	2	Yes
inputregfloat	single precision (32 bit) floating point	input register	2	Yes
holdingregdouble	double precision (64 bit) floating point	holding register	4	Yes
inputregdouble	double precision	input	4	Yes

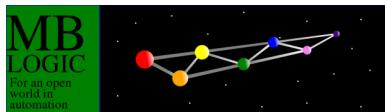


	(64 bit) floating point	register		
holdingregstr8	string	holding register	2 characters per register	Yes
inputregstr8	string	input register	2 characters per register	Yes
holdingregstr16	string	holding register	1 character per register	Yes
inputregstr16	string	input register	1 character per register	Yes

6.2.4.3 Configuration Items

Valid items are:

- addrtype - This may be any valid address type.
- memaddr - This must be a valid data table address.
- datatype - This must be a valid supported HMI protocol address type. See the section on supported data types for more information on which HMI protocol data types are supported.
- range - This is the range limit applied to numeric values. Attempting to write a value outside of this range will result in a "range" error. Range takes two parameters separated by a comma. The first is the lower limit, and the second is the upper limit.
- scale - This is a scale factor which is applied to values written to or read from the data table. This takes two parameters, separated by a comma. The first parameter is the offset (or "b") and the second parameter is the gain (or "m"). When a value is read from the data table and sent to the client, the formula $y=mx + b$, where "x" is the register value and "y" is the value sent to the client. When a value is written to a register, the formula $y=(x - b)/m$ is applied, where



"x" is the value sent from the client, and "y" is the value written to the data table. If the resulting address is out of range, an error is returned.

- `strlen` - This is the maximum length for strings. This is only valid for string data types.

6.2.4.4 Defaults and Type Over-rides

The data table address type is the primary configuration parameter. If the data type can be determined by the address type, then the data type will automatically default to the correct type and any configured data type will be ignored. Any additional parameters which are not required for that address type will also be ignored.

When "addrtype" is "discrete" or "coil":

- The data type is assumed to be "boolean", and any configured "datatype" is ignored.
- Any configured "range", "scale", or "strlen" are ignored.

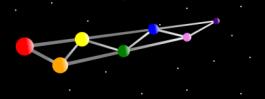
When "addrtype" is a selection corresponding to an integer or floating point storage type:

- If no datatype is configured, a default of "integer" is used.
- If no range is configured, a default range of -32768 to 32767 is used.
- if no scale is specified, a default offset of 0 and a default gain of 1 is used.
- Any configured "strlen" is ignored.

When "addrtype" is a selection corresponding to a string storage type:

- The data type is assumed to be "string", and any other configured "datatype" is ignored.
- Any configured "range" or "scale" are ignored.

Review the documentation on the system data table before deciding which registers, coils, or discrete inputs to use. Remember that some registers are used to hold the coils and discrete inputs and should not be used as regular registers. Remember that some coils are reserved for resetting communications faults.



6.2.5 Configuring Events and Alarms Tags

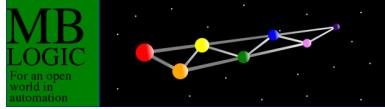
Events and alarms are defined in separate sections. The events section is identified by the name "&events". The alarms section is identified by the name "&alarms". Since HMI protocol tags cannot start with an "&" character, these will not conflict with any other permitted tag name.

Each of the events and alarms sections has three types of items. The "base" item is used to define an offset (or "base address") in the coils area of the data table. The "base" address is added to the address configured for each event or alarm. The events and alarms section must each define a base address. This item takes the form "base=(integer)". For example, "base=1024".

The second type of item is the associate of each event or alarm tag with a data table address. For any item other than "base" the key is assumed to be a data table address, and the value is assumed to be an event or alarm tag. The data table address is added to the "base" value to give the coil address which is monitored for events or alarms.

The third type of data is the zone list. A "zone" is a tag which is used to identify a group of alarms or events. An event or alarm tag can be assigned to any number of zones. The zone list consists of any names following the alarm or even tag. Zone names must be separated by commas.

```
# Event definitions.  
[&events]  
base=32300  
0=PumpRunning, zone3  
1=PumpStopped, zone3  
2=Tank1Empty, zone1  
3=Tank1Full, zone1, zone2  
4=Tank2Empty, zone2  
5=Tank2Full, zone2  
  
# Alarm definitions.  
[&alarms]  
base=32400  
0=PB1Alarm, zone1  
1=PB2Alarm, zone2  
2=PB3Alarm, zone3
```



6.2.6 Configuring the ERP List

The ERP list defines which of the HMI address tags are visible to the ERP protocol. All of the tags listed in the ERP list must be existing address tags.

The ERP list is identified by the section "&erplist". There are two items, "read" and "write". The tags referenced by "read" are available to the "read" ERP command. The tags referenced by "write" are available to the "write" ERP command.

```
[&erplist]
read = PL1, PL4, PumpSpeedActual, PumpSpeedCmd, Tank1Level,
Tank2Number
write = PB1, PumpSpeedCmd
```

6.2.7 Examples

```
# HMI config file.
# 08-Jan-2009.

# Client page(s) version.
[clientversion]
ver=Ver 0.6 Demo

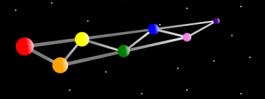
# Server ID.
[serverid]
id=HMI demo server

#####
[PB1]
addrtype=coil
memaddr=0

[PL1]
addrtype=discrete
memaddr=0

[Tank1Level]
addrtype=holdingreg
memaddr=32000
datatype=integer
range=0, 100
scale=0, 1

[PumpSpeedCmd]
addrtype=holdingreg
memaddr=32002
datatype=integer
```



```

range = -1800, 1800
scale = 5, 0.10

[IntegerTagRO2]
addrtype=inputreg
memaddr=40002
datatype=integer
range = -32768, 32767
scale = 0, 1

[FloatTagRW1]
addrtype=holdingreg
memaddr=40001
datatype=float
range = -32768, 32767
scale = 0, 1

#####
# 32 bit integer.
[Testholdingreg32]
addrtype=holdingreg32
memaddr=20000
datatype=integer
range = -2147483648, 2147483647
scale = 0, 1

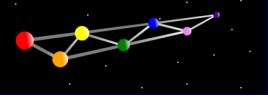
[Testinputreg32]
addrtype=inputreg32
memaddr=20000
datatype=integer
range = -2147483648, 2147483647
scale = 0, 1

# Single precision floating point.
[Testholdingregfloat]
addrtype=holdingregfloat
memaddr=20010
datatype=float
range = -2147483648, 2147483647
scale = 0, 1

[Testinputregfloat]
addrtype=inputregfloat
memaddr=20010
datatype=float
range = -2147483648, 2147483647
scale = 0, 1

# Double precision floating point.
[Testholdinggregdouble]
addrtype=holdinggregdouble
memaddr=20020
datatype=float
range = -2147483648, 2147483647

```



```

scale = 0, 1

[Testinputregdouble]
addrtype=inputregdouble
memaddr=20020
datatype=float
range = -2147483648, 2147483647
scale = 0, 1

# String with 2 characters per register.
# The 'datatype' parameter can be omitted.
[Testholdingregstr8]
addrtype=holdingregstr8
memaddr=20030
datatype=string
strlen=8

[Testinputregstr8]
addrtype=inputregstr8
memaddr=20030
datatype=string
strlen=8

# String with one character per register.
# The 'datatype' parameter can be omitted.
[Testholdingregstr16]
addrtype=holdingregstr16
memaddr=20040
datatype=string
strlen=10

[Testinputregstr16]
addrtype=inputregstr16
memaddr=20040
datatype=string
strlen=9

#####
# Event definitions.
[&events]
base=32300
0=PumpRunning, zone3
1=PumpStopped, zone3
2=Tank1Empty, zone1
3=Tank1Full, zone1, zone2
4=Tank2Empty, zone2
5=Tank2Full, zone2

# Alarm definitions.
[&alarms]
base=32400
0=PB1Alarm, zone1
1=PB2Alarm, zone2
2=PB3Alarm, zone3

```



6.3 Alarm and Event History

6.3.1 Overview

The alarm and event history database can be queried via the provided web based history query system. Most of the help information presented here is also in the alarm and event history page itself under the "help" tab.

6.3.2 Accessing the Alarm and Event History Page

The alarm and event history page is part of the HMI system and is accessed via the same URL as the HMI web pages. This means the HMI server must be configured and running to access the history system. The history system can also operate via the read-only restricted HMI service.

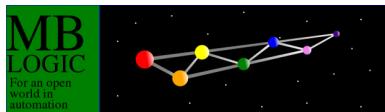
The history web page will be listed as one of the "HMI Web Pages" on the "System Configure" web page in the status system. With the default configuration provided with the system, the history web page can be accessed directly via the following URL:

```
http://localhost:8082/history-en.html
```

You may need to change the IP number ("localhost" in this example) or port (8082 in this example) depending on how you have installed the system.

6.3.3 Event History

The event database can be queried for events by date, by event name, or by a combination of the two. Select the query method you wish to use by selecting the appropriate check box.



Alarm and Event History - Minefield

File Edit View History Bookmarks Tools Help

Alarm and Event History <http://localhost:8082/history-en.html>

Demo Manufacturing Inc. MBLogic for an open world in automation

Events Alarms Help

Event Query

By Date: By Event:

Date:

From: Year: Month: Day: Hour: Minute: Second:

To: Year: Month: Day: Hour: Minute: Second:

Events:

[Tank 1 is empty.] OR [Tank 2 is empty.]

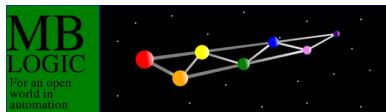
Time	Event
Fri 17 Dec 2010 12:44:44 PM EST	Tank 1 is empty.
Fri 17 Dec 2010 01:52:53 PM EST	Tank 2 is empty.
Fri 17 Dec 2010 04:31:07 PM EST	Tank 1 is empty.
Fri 17 Dec 2010 07:28:28 PM EST	Tank 2 is empty.
Fri 17 Dec 2010 07:54:23 PM EST	Tank 1 is empty.
Fri 17 Dec 2010 07:59:32 PM EST	Tank 1 is empty.

6.3.3.1 Querying Events by Date

To query events by date, enter the desired date range in the date form. The resulting query will fetch events which occurred between (and including) those two dates. Pressing the "reset" button will reset the date range to the default values.

6.3.3.2 Querying Events by Event Name

To query events by event name, select the desire events from the drop down selection widget. Each time you select an event, it will be added to the query and you will see it listed below the selection widget. To clear the current selection, press the "reset" button.



When you select multiple events, the request is "OR"ed together. That is, the query match record which contain any of the events requested.

6.3.3.3 Querying by Both Date and Event Name

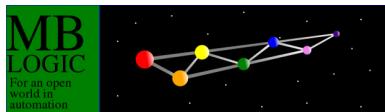
You can query by a combination of both date and event name by selecting both options at once. This will select records which are both within the requested date range and which contain the requested events.

6.3.3.4 Starting the Query

Once you have specified the search criteria, press the "Get Events" button. This will send the request to the system and display the results when they are returned. The results will be displayed in the table beneath the "Get Events" button. The results will show the date at which the event occurred, and the name of the event.

6.3.4 Alarm History Query

The alarm history database can be queried for events by date, by alarm name, by count, or by a combination of any two or three these. Select the query method you wish to use by selecting the appropriate check box.



Alarm and Event History - Minefield

File Edit View History Bookmarks Tools Help

Alarm and Event History <http://localhost:8082/history-en.html>

Demo Manufacturing Inc. **MBLogic** for an open world in automation

Events Alarms Help

Alarm Query

By Date: By Alarm: By Count:

Date:

From: Year: Month: Day: Hour: Minute: Second:

To: Year: Month: Day: Hour: Minute: Second:

Alarms:

[PB1 was pressed.] OR [PB3 was pressed.]

Count: From: To:

Time	Time OK	Alarm	Client	Count
Mon 20 Dec 2010 12:00:41 AM EST	Mon 20 Dec 2010 12:00:50 AM EST	PB3 was pressed.	HMI Demo	3
Mon 20 Dec 2010 12:00:37 AM EST	Mon 20 Dec 2010 12:00:56 AM EST	PB1 was pressed.	HMI Demo	3
Mon 20 Dec 2010 12:01:06 AM EST	Mon 20 Dec 2010 12:01:08 AM EST	PB1 was pressed.	HMI Demo	1
Mon 20 Dec 2010 12:01:23 AM EST	Mon 20 Dec 2010 12:01:28 AM EST	PB3 was pressed.	HMI Demo	2

6.3.4.1 Querying Alarm History by Date

To query alarm history by date, enter the desired date range in the date form. The resulting query will fetch alarms which occurred between (and including) those two dates. Pressing the "reset" button will reset the date range to the default values.

6.3.4.2 Querying Alarm History by Alarm Name

To query alarm history by alarm name, select the desire alarms from the drop down selection widget. Each time you select an alarm, it will be added to the query and you will see it listed below the selection widget. To clear the current selection, press the "reset" button.



When you select multiple alarms, the request is "OR"ed together. That is, the query match record which contain any of the alarms requested.

6.3.4.3 Querying Alarm History by Count

To query alarm history by count (the number of times the alarm occurred before being sent to the history), enter the desired count range in the count form. The resulting query will fetch alarms which occured between (and including) those two count ranges.

6.3.4.4 Querying by Multiple Criteria

You can query by any combination of date, alarm name, and count by selecting multiple options at once. This will select records which meet all the selected criteria. Criteria which are not selected will not factor in the search.

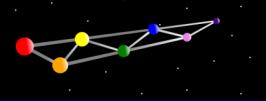
6.3.4.5 Starting the Query

Once you have specified the search criteria, press the "Get Alarms" button. This will send the request to the system and display the results when they are returned. The results will be displayed in the table beneath the "Get Alarms" button. The results will show the following:

- The date at which the alarm originally occurred.
- The date at which the alarm became "ok".
- The name of the alarm.
- The name of the client which acknowledged the alarm.
- The number of times the alarm occurred before being sent to history.

6.3.5 Alarm and Event Messages

The alarm and event history must be integrated with the HMI application in order to use the same alarm and event messages. These messages are read from a file. The default name for this file is *messagetexts.js*. The alarm texts are in an object called *MBT_AlarmText*. The event texts are in an object called *MBT_EventText*.



```
MBT_AlarmText = {  
    "PB1Alarm" : "PB1 was pressed.",  
    "PB2Alarm" : "PB2 was pressed.",  
    "PB3Alarm" : "PB3 was pressed.",  
    "PB4Alarm" : "PB4 was pressed."  
};  
  
MBT_EventText = {  
    "PumpRunning" : "Tank pump is running.",  
    "PumpStopped" : "Tank pump is stopped.",  
    "Tank1Empty" : "Tank 1 is empty.",  
    "Tank1Full" : "Tank 1 is full.",  
    "Tank2Empty" : "Tank 2 is empty.",  
    "Tank2Full" : "Tank 2 is full."  
};
```

If your HMI application uses a different file to store the messages, or different object names, you will need to change the alarm and event history web page accordingly.

6.3.6 System Components

The alarm and event history system draws its data from the alarm and event database. Alarm and event logging is described elsewhere in this documentation and so is not covered in any more detail here.

The alarm and event history reporting system consists of the following components:

- **history-en.html** - This is a web application contained in a web page. This page can be renamed if desired.
- **history.css** - This is a text file which contains information about styles such as colours, fonts, text sizes, etc. This is a standard web page style sheet and information about it can be found in any reference about creating web pages.
- **pagebackground.png** - This is an image file which provides a page background.
- **pageicon.png** - This is an image file which provides a web page icon.
- **pagelogo.png** - This is an image file which provides a web page image.
- **Messages** - These are the actual message texts which convert the alarm and event "tags" to readable messages. See the above section on "Alarm and Event Messages" for more information.



- **Javascript libraries** - These are files containing Javascript code which run the alarm and event history web page application. These are *libhistory.js* and *libdata.js*.

With the exception of the Javascript libraries, all history components must be located in the **hmipages** directory so they can be served by the HMI web server. The Javascript library files are located with the other standard HMI Javascript libraries in another directly within the system software.

6.3.7 Customising the Alarm and Event History

The alarm and event history page has various page decorations which can be changed to match the desired theme or appearance of the control application. They can be changed by editing the web page with a text editor. These consist of the following:

- Logo
- Page Icon
- Background
- Page Heading

6.3.7.1 Logo

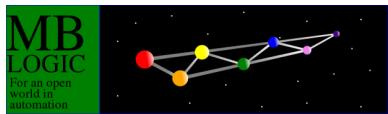
The logo is an image found at the top left of the page. The default logo used is called *pagelogo.png*.

6.3.7.2 Icon

The page icon is a small image which is used by various web browsers in different ways. For example, with Firefox it appears in the page tab, as well as to the left of the URL bar, and also in the bookmarks. The default icon used is called *pageicon.png*.

6.3.7.3 Background Image

The background image appears around the edges and at the bottom of the web page. The default background image used is called *pagebackground.png*.



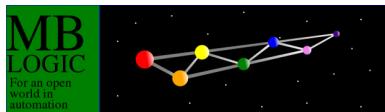
The logo, page icon, and background image are the same files as used with the HMI demo application. They can be changed to use different file names, different files with the same names can be substituted, or they can be removed altogether. The logo and icon file names are specified in the HTML file, while the page background is specified in the CSS file.

6.3.7.4 Page Heading

The page heading is a text message which appears at the top of the page to the right of the logo. This is located in the HTML file and can be changed to suit the application.

6.3.7.5 CSS Styles

In addition to these, the CSS file can be modified to change the colours, fonts, and other features. The HTML file itself can be changed as desired.



6.4 HMIBuilder

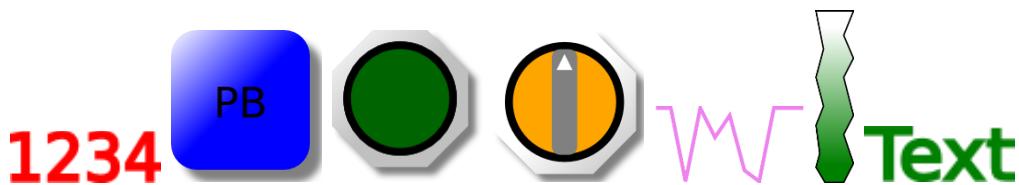
6.4.1 Overview

MBLogic HMIBuilder is a program which allows you to construct web based HMI systems using simple editing techniques. A set of templates is provided which you can customise or use as is. Graphical HMI screens are constructed using the Inkscape drawing editor (a popular free drawing editor). MBLogic HMIBuilder will combine the Inkscape drawings with the templates to produce a finished HMI web page.

The following is a brief summary of what can be done with MBLogic HMIBuilder. Complete documentation is included with HMIBuilder.

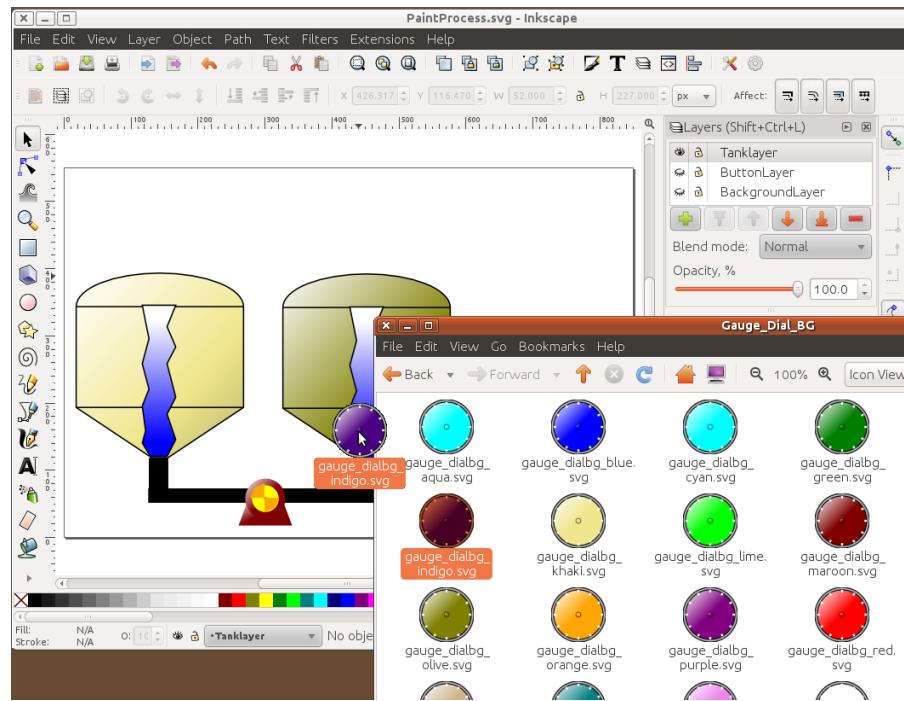
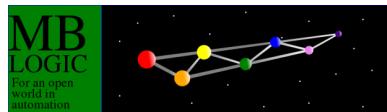
6.4.2 HMIBuilder Widget Library

MBLogic HMIBuilder comes with a selection of HMI widgets (controls) which can be dragged into a drawing and positioned visually. A large library of HMI widgets is provided with HMIBuilder, and custom widgets can be constructed if desired.



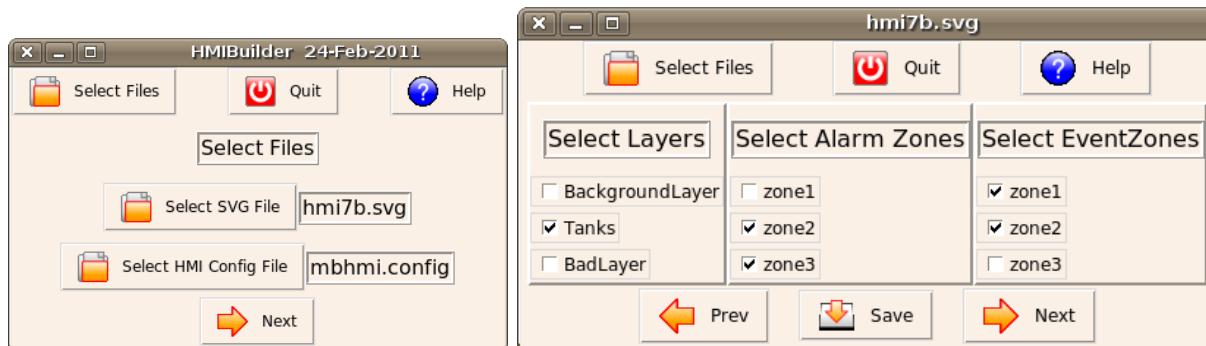
6.4.3 Inkscape Drawing Editor

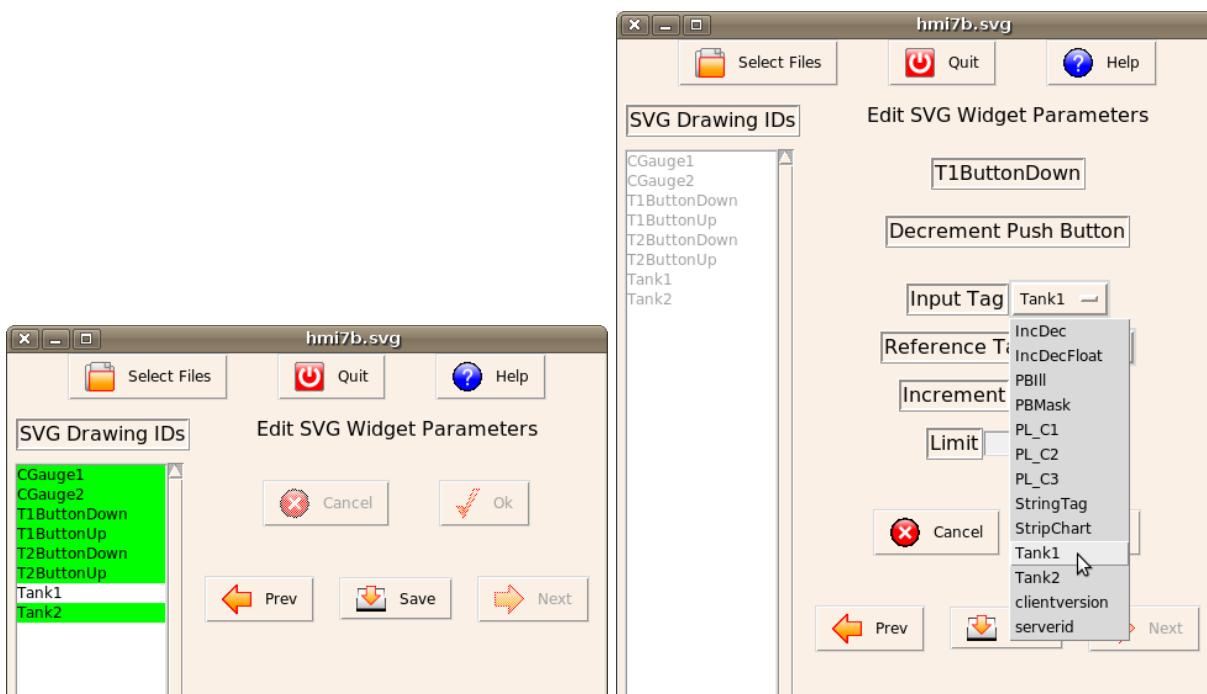
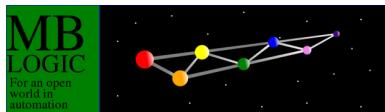
Inkscape is a free open source drawing editor which is widely used to make professional grade drawings. The native drawing format for Inkscape is SVG (Scalable Vector Graphics) which is also the standard native vector drawing format for web pages. MBLogic HMIBuilder HMI widgets can be dragged onto an Inkscape drawing and positioned visually. In addition, custom art work can be created using Inkscape's drawing capabilities.



6.4.4 MBLogic HMIBuilder

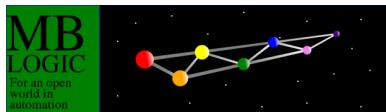
MBLogic HMIBuilder will take drawings assembled using Inkscape and automatically generate the scripting and data and insert it into a web page template to produce a finished HMI web page without programming. Options are selected using drop down menus and check boxes.





6.4.5 Help for HMIBuilder

MBLogic HMIBuilder comes with a complete set of help files which will appear in your web browser when you click on the "help" button. The same help files are also available at the MBLogic project web site.



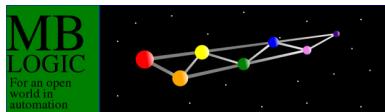
6.5 HMI Client Libraries

6.5.1 Overview

The HMI Client Libraries are Javascript libraries which are included in HMI web pages to enable continuous communications with the HMI server, and to enable control of the input and output HMI widget devices. The web page templates provided automatically import these libraries, but if you create your own web page templates you will need to import these libraries yourself.

6.5.1.1 *Libraries*

- [HMI Client Protocol Library](#) - This library is responsible for handling all communications between the HMI web browser (client) and the HMI server. In addition, it maintains the "display list" which is responsible for calling the display objects and updating them with new data as it becomes available. In addition, input device widgets will use functions in this library to send their data to the server.
- [HMI Client Display Library](#) - The client display library provides a series of code objects which are used to update the web browser SVG to reflect the state of the protocol "tags" they are monitoring. The protocol library will scan through the list of code objects in use and update them whenever new data arrives. Each library object performs a specific function which emulates such things as pilot lights, numeric outputs, gauge readings, etc. These functions are named in a manner which reflects their use in such familiar devices. However, their use is not limited to the provided SVG clip-art and user created clip art can also be animated with these functions.
- [HMI Client Event and Alarm Library](#) - The client even and alarm library is used to display alarms and events in HTML tables. The web page templates already come with the necessary HTML structures and calls to the libraries to handle alarms and events. User created custom web pages can also make use of this library.



6.5.2 HMI Client Protocol Library

6.5.2.1 Overview

The HMI protocol client Javascript library comes in two parts. These are the JSON encoding library and the HMI protocol library itself.

6.5.2.2 JSON and the HMI System

JSON is a standard data encoding format used in web applications. The HMI messages are encoded in JSON format. The HMI system can use standard JSON encoding libraries such as "json2.js" from [JSON.ORG](#). With the increasing popularity of JSON, future versions of web browsers are expected to support it natively without requiring a separate Javascript library.

6.5.2.3 HMI Communications Library

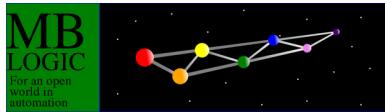
The HMI communications library is implemented as a single Javascript object. This library ("libhmiclient2.js") does more than just support the HMI protocol. It provides the following features:

- Encoding and decoding of messages using the HMI protocol.
 - Buffered HMI reading and writing functions.
 - Event and alarm handling.
 - Automatic update of screen displays.
-

6.5.2.4 The Read List

The "read list" is an array of tag names that are to be polled (read) on a regular basis. The "read list" is a critical part of an application, because it defines what data is available to the client for display.

Example -



```
// Make a list of all the address tags to be monitored. This is what  
we  
// send to the server asking for values.  
var ReadList = ["PL1", "PL2"];
```

6.5.2.5 The Zone List

The "zone list" is an array of zone names that defines the alarms and events that are to be monitored.

```
Example -  
  
// Make a list of the alarm and event zones to be monitored. Zones are  
// used to filter alarms and events to only those we are interested  
in.  
var AlarmZoneList = ["zone1", "zone2", "zone3"];  
var EventZoneList = ["zone1", "zone2", "zone3"];
```

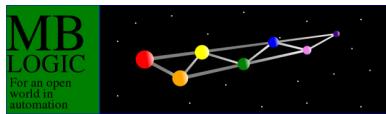
6.5.2.6 The Display List

The display list is an array of code objects which control graphics, text, tables, etc. "UpdateDisplay" calls each item in turn, passes it the latest data, and asks it to update itself. Each screen item must be a code object which exports a method called "UpdateScreen", and which accepts one parameter. e.g. "objref.UpdateScreen(presentstate);"

To add an item to the display list, create the object and pass it to "AddToDisplayList".

```
For example -  
  
// This defines a pilot light control.  
var PL1 = new MB_PilotLight("green", "red", "black", "PL1",  
document);  
MBHMIProtocol.AddToDisplayList(PL1, "PL1", "read");
```

This creates a pilot light using "MB_PilotLight" (part of another library), assigns it to a reference called PL1, and passes it to "AddToDisplayList" along with two other parameters.



The second parameter is the "address label". This is used by instructions which need to read the input values. This should be a valid tag from the read list whenever the address label is "read". Its value is not significant for other address labels.

The third parameter is the "tag type". This is a description of the type of address used in the second parameter. This determines the source of the data which is passed to the display item. Valid values are: "read" (input data from the read list), "timestamp" (the server time stamp from the last message), "stat" (the server stat value from the last message), "msgid" (the message id returned by the server), "events" (an array of the latest event messages), and "alarms" (an object containing alarms and alarm acknowledgements).

Alarms and alarm acknowledgements are contained in a single object in order to fit both within the single parameter limit. This object has the following form : {"alarms" : this._AlarmBuffer, "alarmsupdated" : this._AlarmsUpdated, "alarmack" : this._AlarmAckBuffer, "alarmackupdated" : this._AlarmAckUpdated};

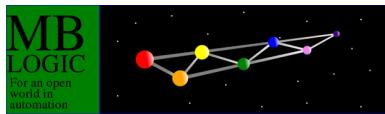
In this example, "alarms" is the alarm input buffer. "alarmsupdated" is an integer that increments whenever new alarms have arrived. "alarmack" is the alarm acknowledgement input buffer. "alarmackupdated" is an integer that increments whenever new acknowledgement messages have arrived.

6.5.2.7 *Initialisation*

6.5.2.7.1 `HMIClient(url, port, cid, readlist, alarmzonelist, eventzonelist, ansyncoption);`

This is the object name and the initialisation parameters.

- `url` = The url (address) of the server, including address and path.
- `port` = The IP port of the server.
- `cid` = The client ID string.
- `readlist` = The IO read list.
- `alarmzonelist` (array) - List of alarm zone filters.
- `eventzonelist` (array) - List of event zone filters.
- `ansyncoption` = If true, uses asynchronous communication. If false, uses synchronous communications.



The web browser will enforce a security policy requiring that the communications between the HMI web page and the server must be to the same host (address) and port as the web page was loaded from. This means the communications library needs to know the host name and port. This can be accomplished in several ways. One way is to hard code these values. However, that means they must be known in advance.

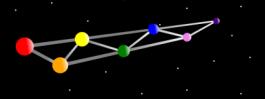
The other way is to use a standard Javascript object to obtain these values automatically. These are:

- **window.location.hostname** - This automatically obtains the host name (address).
- **window.location.port** - This automatically obtains the port number.

You must also decide whether to use synchronous or asynchronous communications. With synchronous communications, the web browser waits for the server to reply before continuing. With asynchronous communications, the web browser does not wait. Rather, it sends the request and continues running the browser script. When the response from the server is received, the software function which handles the response is automatically called and the display list is updated.

When the web browser and the server are communicating on a reliable local network (or on the same computer), there is normally no noticeable difference between the two methods. However, if the network is slow or unreliable, using the synchronous method can cause the web browser to appear to "stall" or be unresponsive. Also, if the server is stopped and restarted for some reason, the web page will require a reload (press "F5" or click on the "reload" icon) before communications will resume.

When asynchronous communications is used, these problems will not occur. The communications library will automatically keep retrying communications. However, some versions of some web browsers when running on some operating systems may have bugs or other problems which prevent asynchronous communications from working correctly. This rarely occurs. However, since applications using asynchronous communications are relatively new, this feature is not as well tested in all browsers as the synchronous method. If problems are encountered when using asynchronous communications, it is suggested to try setting the "ansyncoption" parameter to "false" to see if the synchronous method will work instead.



```

// Make a list of all the address tags to be monitored. This is what
we
// send to the server asking for values.
var ReadList = ["PL1", "PL2"];

// Make a list of the alarm and event zones to be monitored. Zones are
// used to filter alarms and events to only those we are interested
in.
var AlarmZoneList = ["zone1", "zone2", "zone3"];
var EventZoneList = ["zone1", "zone2", "zone3"];

/* This handles communications with the server.
The parameters are:
1) The host name the web page was loaded from.
2) The port number the web page was loaded from.
3) The client ID string.
4) The list of tags to poll for data.
5) The list of alarm zones to poll for new alarms.
6) The list of event zones to poll for new events.
7) true = Enable asynchronous communications.
The first two parameters use a standard Javascript feature.
Alternatively, these can be hard coded values if the host
and port are known in advance.
*/
var MBHMIProtocol = new HMIClient(window.location.hostname,
                                  window.location.port, "Any client name
string.",
                                  ReadList, AlarmZoneList, EventZoneList, true);

```

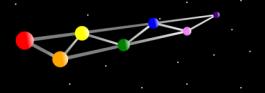
There should only be **one** HMI communications object in an application that takes care of **all** polling.

The "same origin" security policy in most web browsers requires that a web client application can only communicate with the same host that the web page was loaded from. This means that a single server must be used as the source of the web page and as the source (and destination) of all data.

6.5.2.8 Updating the Read List Dynamically

6.5.2.8.1 SetReadList(readlist)

- **readlist: (array)** = An array of strings containing the list of address tags to read from the server.



The read list can be updated while the system is running.

```
MBHMIProtocol.SetReadList (NewReadList) ;
```

This will change the read list that was set as part of the initialisation process. This is *not* normally necessary. It *may* be useful in reducing the amount of data polled in some larger applications. If you are not sure whether you need this feature, you probably do not need it.

6.5.2.9 *Reading and Writing*

6.5.2.9.1 AddWrite(writetag, writeval)

- **writetag** = The tag to write to.
- **writeval** = The value to write
- Adds a write request to the output queue.

```
MBHMIProtocol.AddWrite ("PumpSpeedCmd", PumpSpeed) ;
```

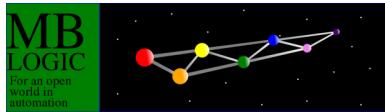
6.5.2.9.2 WriteImmediate(writetag, writeval)

- **writetag** = The tag to write to.
- **writeval** = The value to write.
- Similar to AddWrite, but instead of simply adding a request to the write queue, it forces an immediate polling cycle and screen update.

```
// Stop the pump.  
if (ButName == "PBStop") {  
    PumpSpeed = 0;  
}  
// Update the new pump speed.  
MBHMIProtocol.WriteImmediate ("PumpSpeedCmd", PumpSpeed) ;
```

6.5.2.9.3 WriteToggleImmediate(writetag, reftag)

- **writetag** = The tag to write to.
- **reftag** = The reference tag.



- Similar to WriteImmediate, but instead of simply writing a value, it sets the value of a tag to the boolean opposite of that found in a reference tag.

```
MBHMIProtocol.WriteToggleImmediate('PB2', 'PL2');
```

6.5.2.9.4 WriteIncImmediate(writetag, reftag, incval, inclimit)

- writetag = The tag to write to.
- reftag = The reference tag.
- incval = The amount to increment by. This must be a numeric value. This may be negative.
- inclimit = The absolute value of the increment limit. This must be a numeric value.
- Similar to WriteImmediate, but instead of simply writing a value, it reads the value stored in "reftag", increments it by the number "incval", and stores the result in "writetag". "reftag" and "writetag" may be the same. If the absolute value of the result exceeds the number "inclimit", the result is set to zero.

```
MBHMIProtocol.WriteIncImmediate('PB4', 'PL4', 1, 6);
```

6.5.2.9.5 function GetServerID()

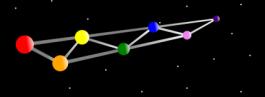
Read the value of the id string.

6.5.2.9.6 function GetMsgStat()

Read the value of the current message status.

6.5.2.9.7 GetRead(readlabel)

- readlabel = The tag to read.
- Reads a value from tag address "readlabel". The value returned is the most recent value in the input buffer.



```
var PumpSpeed;
// First, get the current pump speed.
PumpSpeed = MBHMIProtocol.GetRead("PumpSpeedActual");
```

6.5.2.9.8 GetTimeStamp()

- This returns the most recent server time stamp. The value returned is the one from the most recent message.

```
var TimeStamp = MBHMIProtocol.GetTimeStamp();
```

6.5.2.10 Event and Alarm Handling

6.5.2.10.1 AddAlarmAck()

- This adds the alarm at the head of the alarm queue (the oldest unacknowledged alarm) to the output queue.

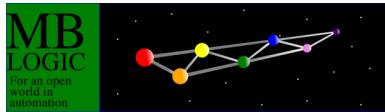
```
MBHMIProtocol.AddAlarmAck();
```

6.5.2.11 Display Control

6.5.2.11.1 AddToDisplayList(objref, addrlabel, tagtype)

- objref = A reference to the display object.
- addrlabel = If the tagtype is "read", this is the address label the display object is attached to. If the tagtype is not "read", the value in this parameter is ignored.
- tagtype = The type of tag.

This adds an HMI display Javascript object to the display list. The "addrlabel" must be a valid tag which is part of the read list. The "tagtype" is a string, and may be "read", "timestamp", etc. (see below). Each of these tag types refers to a valid HMI field from which it derives its data.



The display list is automatically updated whenever new data is received from the server.

```
// This defines a pilot light control.  
var PL1 = new MB_PilotLight(document, "PL1", "black", "green", "red");  
var PL2 = new MB_PilotLight(document, "PL2", "black", "green", "red");  
  
// Now, add each of these screen objects to the list of things to  
update.  
MBHMIProtocol.AddToDisplayList(PL1, "PL1", "read");  
MBHMIProtocol.AddToDisplayList(PL2, "PL2", "read");  
  
// This is for the alarm status display on the main screen.  
var AlarmStat = new MB_PLMultiColour(document, "AlarmStat",  
    "black", ["white", "green", "orange", "red"]);  
MBHMIProtocol.AddToDisplayList(AlarmStat, "", "unackalarm");
```

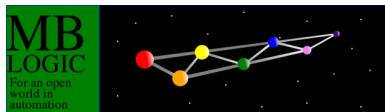
6.5.2.11.2 HMI Tag Types

- **read** - Use the data from the tag given as the second parameter.
- **timestamp** - The message time stamp.
- **serverid** - The server id.
- **stat** - The message "stat" parameter.
- **msgid** - The message id number.
- **alarms** - The alarms.
- **alarmhistory** - The alarm history.
- **events** - The events.
- **errors** - The errors.
- **status** - The message status.
- **chkalarm** - Check if alarms present (see below).

For "chkalarm" alarms, the HMI display object will be passed the following values:

- 0 - No current alarms present.
- 1 - At least one unacknowledged alarm in the "ok" state.
- 2 - At least one acknowledged alarm is still active ("ackalarm").
- 3 - At least one unacknowledged alarm in the "alarm" state.

Higher number codes take priority over lower numbered codes.



6.5.2.12 Polling Cycle Control

6.5.2.12.1 SendRequest()

This triggers a communications polling cycle. However, it does not *schedule* the polling request. Scheduling the polling request must be done by other means. Typically, you would use "setTimeout", which is a standard web page Javascript event. In the example below, the function "RunScanCycle" is being scheduled again in 1000 milliseconds.

The polling time to use will be a compromise between several factors. If the time between polls is too long, there will be a noticeable lag between a physical event happening and indication of it on the display. If the time between polls is too short, the load on the computer displaying the HMI client (and possibly the server as well) will be too high.

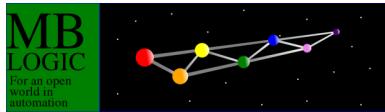
Also, if asynchronous polling is used, and if the time between polls is too short, the server will not have had enough time to respond before the client begins to send the next poll. If this happens, the connection will be aborted and the client will not receive the reply. In these cases, it is better to either use synchronous communications (this is set via an initialisation parameter) or to use a slower polling rate.

```
// Run all the operations required each scan cycle.  
function RunScanCycle() {  
  
    // Query the server for updates.  
    MBHMIProtocol.SendRequest();  
  
    // Call the function back again at the set interval.  
    window.setTimeout("RunScanCycle()", 1000);  
}
```

6.5.2.13 Communications Watchdog

6.5.2.13.1 CommsWatchDogTimeOut(limit)

- limit = The amount of difference permitted between the number of messages sent and received before an error is reported.



This function is useful only when using asynchronous communications. The "SendRequest" function keeps track of the number of requests sent and received. If too many consecutive requests are sent without a response from the server, "CommsWatchDogTimeOut" will return "true". This may be used to annunciate a loss of communications on the screen. The count of requests is automatically reset every time a response is received, so an occasional loss of messages will automatically correct itself when the next response is received.

If communications is lost, a valid response message will **not** be received, which in turn means the display list will not be called to update itself. If you wish to display a message to the operator, you should call the appropriate function directly yourself. If you are using one of the client display library functions, you may do this by called the "UpdateScreen") method directly.

This method is an alternative to using the watchdog list. The watchdog list method using "SetCommsWatchDogTimeOut" and "AddToWatchdogList" is recommended for most normal applications.

```
// Run all the operations required each scan cycle.
function RunScanCycle() {

    // Query the server for updates.
    MBHMIProtocol.SendRequest();

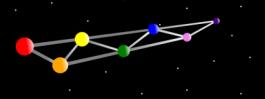
    // Check the comms watch dog counters.
    ComsTimeOut = MBHMIProtocol.CommsWatchDogTimeOut(10);
    if (ComsTimeOut) {
        CommWD.UpdateScreen(1);
    } else {
        CommWD.UpdateScreen(0);
    }

    // Call the function back again at the set interval.
    window.setTimeout("RunScanCycle()", 1000);

}
```

6.5.2.13.2 SetCommsWatchDogTimeOut(limit)

- **limit = limit (integer)** - The amount of difference permitted between the number of messages sent and received before an error is reported. This is a count, not a time value.



Set the limit to be used when checking the communications watchdog limit via the watchdog list. The watchdog function checks if responses are being received from the server by comparing a communications counter to the limit.

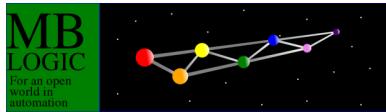
```
// Set the watchdog limit to 10 scans.  
MBHMIProtocol.SetCommsWatchDogTimeOut(10);
```

6.5.2.13.3 AddToWatchdogList(objref)

- **objref** = A reference to the display object.

Add an item to the watchdog list. This is similar to the "display list" but is only for the communications watchdog counter. Objects on this list are scanned and automatically updated each time a request is sent. The watchdog function passes a 1 to the display function if the communications watchdog counter has exceeded the limit, or 0 otherwise.

```
// This is for the communications watch dog display.  
var CommWD = new MB_PilotLight(document, "CommWatchDog", "black",  
"green", "red");  
// This is the watchdog list, not the regular display list.  
MBHMIProtocol.AddToWatchdogList(CommWD);
```



6.5.3 HMI Client Display Library

6.5.3.1 Overview

The HMI system is based on open standards such as JSON, Javascript, SVG, and HTML. The HMI client display library is not required to build a web based HMI which can interact with the HMI server. However, the HMI client display library provides a number of useful functions which make creating web based HMI systems easier. These functions are documented below.

The client display library is implemented as a series of independent objects. Most of these objects are designed to be compatible with the HMI communications library "display list".

6.5.3.2 Libraries

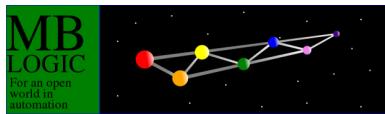
The client display library consists of the following:

6.5.3.3 Current HMI Libraries

- *libmbhmicontrols2.js* - This provides control for most of the basic HMI output widgets such as pilot lights, selector switches, text and numeric displays, etc.
- *libmbhmimisc.js* - This provides control for additional HMI features such as selecting screens, numeric keypad input, strip charts, etc.
- *libmbhmiold.js* - This provides control for obsolete HMI features and should only be used to provide compatibility for older HMI applications. This contains the features from the original library which are now considered obsolete.

6.5.3.3.1 Obsolete HMI Libraries

- *libmbhmi.js* - This is the original library and is now obsolete. It is only present to provide support for older HMI applications. New applications should not include this file. Not all functions present in the newer libraries are present in this old one. This library should not be used in the same web page as the newer versions as it duplicates many of the same functions.



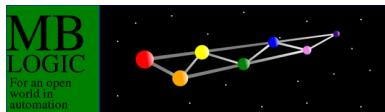
- *libmbhmicontrols.js* - This is an older version of the libmbhmicontrols2.js library. It does not work with web pages created using the HMIBuilder library, nor with the new widget sets. You can continue to use this with older HMI web pages. However, you cannot use both *libmbhmicontrols.js* and *libmbhmicontrols2.js* in the same web page as they will conflict with each other. Import one or the other into a web page, but not both at the same time.
-

6.5.3.4 Function Summary

The following is a short summary of SVG properties which can be controlled, and the functions provided to manipulate them. See the individual functions for details on how each one operates.

6.5.3.4.1 libmbhmicontrols2.js

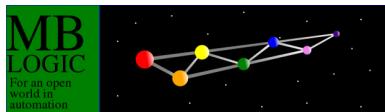
Function	SVG Property	Used For
MB_PilotLight	fill	Two colour pilot lights.
MB_PLMultiColour	fill	Pilot lights with multiple colours.
MB_PilotLightStat	fill	Two colour pilot lights which need to be compared to a status variable.
MB_PilotLightFlashTransform	animateTransform	Pilot lights using an animated flash.
MB_2PosSSDisplay	rotate	Rotation angle display for 2 position selector switch.



MB_3PosSSDisplay	rotate	Rotation angle display for 3 position selector switch.
MB_NumericDisplay	text value	Display numeric values.
MB_NumericFloatDisplay	text value	Display numeric values with a specified number of decimal places.
MB_StringDisplay	text value	Display text strings.
MB_TextDisplay	text value	Display a text message selected from a list of messages.
MB_DialGauge	rotate	Rotate the pointer on a dial gauge.
MB_ColumnGauge	colour gradient stops	Display the level in a column using colour gradients.
MB_Pipe2	fill	Change the colour of a pipe (similar to a pilot light).
MB_PumpRotate	rotate	Rotate a pump.
MB_PumpRotateAnimated	animateTransform	Rotate a pump using animations.

6.5.3.4.2 libmbhmimisc.js

Function	SVG Property	Used For

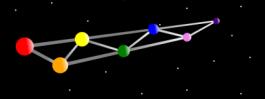


MB_ScreenSelect	Display style - block or none	Select one HMI screen from several
MB_NumericPad	N/A	Enter numeric data
MB_SlideDisplay	transform - translate	Move an SVG graphic on the screen
MB_StripChart	polyline values	Update a strip chart line
MB_GraphicSelect	Display style - block or none	Select one SVG graphic from among several
MB_GraphicShowHide	Display style - block or none	Hide or show an SVG graphic

6.5.3.4.3 libmbhmiold.js

The following are briefly described to help understand existing HMI applications. They should not be used in new applications.

Function	SVG Property	Used For	Replace With
MB_SVGPushButton	colour gradient stops	Provides a colour effect when a push button is pressed	Push buttons using CSS styles to provide colour effects.
MB_Pipe	stroke	Changes the colours of the walls of a pipe	MB_Pipe2



MB_PipeFlow	colour gradient stops	Provides the effect of "bubbles" moving along a pipe	MB_Pipe2
MB_PipeFlow2	colour gradient stops	Like MB_PipeFlow but implemented using a different method	MB_Pipe2
MB_LEDDigit	Display style - block or none	Used to select one seven segment style digit from among all 10 possible digits	Use a regular numeric display.

6.5.3.5 *Pilot Lights*

Pilot lights operate by changing the "fill" attribute of a graphic. Any SVG graphic which has a "fill" attribute can act as a pilot light.

6.5.3.5.1 MB_PilotLight(svgdoc, PLID, initcolour, offcolour, oncolour)

Pilot light control. This changes the "fill" property to indicate one of two states.

- svgdoc - Reference to SVG document.
- PLID (string) - ID of SVG item.
- initcolour (string) - Colour to use when undefined.
- offcolour (string) - Colour to use when off.
- oncolour (string) - Colour to use when on.

```
var PL1 = new MB_PilotLight(document, "PL1", "black", "green", "red");
```

6.5.3.5.2 MB_PLMultiColour(svgdoc, PLID, initcolour, colourlist)

Pilot light control with multiple colours. This selects between a list of colours, depending on the value of the integer at the tag address. The colours must be defined



in an array of strings. This array determines what colour is displayed for each integer value. Unlike MB_TextDisplay, this function does not display the text string itself.

- `svgdoc` - Reference to SVG document.
- `PLID (string)` - ID of SVG item.
- `initcolour (string)` - Colour to use when undefined.
- `colourlist (array)` - Array of strings containing colours.

```
var ColourList = ["red", "orange", "yellow", "green", "blue",
"indigo", "violet"];
var PL4 = new MB_PLMultiColour(document, "PL4", "black", ColourList);
```

6.5.3.5.3 MB_PilotLightStat(svgdoc, PLID, initcolour, offcolour, oncolour, okstat)

Pilot light control. If the present value is equal to the comparison value, the light will turn "off", otherwise, it will turn "on". Any value not equal to the comparison value is considered to be the result of a condition which must be indicated.

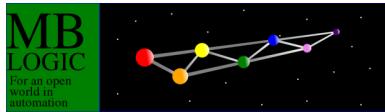
- `svgdoc` - Reference to SVG document.
- `PLID (string)` - ID of SVG item.
- `initcolour (string)` - Colour to use when undefined.
- `offcolour (string)` - Colour to use when off.
- `oncolour (string)` - Colour to use when on.
- `okstat (string)` - Value to compare to for "ok". This should be a constant.

```
// This is for the server status display on the test error screen.
var ServerStat2 = new MB_PilotLightStat(document, "ServerStat2",
"black",
"green", "red", "ok");
MBHMIProtocol.AddToDisplayList(ServerStat2, "stat", "stat");
```

6.5.3.5.4 MB_PilotLightFlashTransform(svgdoc, PLID)

Flashing pilot light control. The flashing does not depend on the scan rate. This uses the SVG `animateTransform` property.

- `svgdoc` - Reference to SVG document.
- `PLID (string)` - ID of SVG item.



```
var PLAN1 = new MB_PilotLightFlashTransform(document, "PLAN1");
```

6.5.3.6 Selector Switches

Selector switches rotate according to the state or value of a tag. These functions provide an **output** indication. Data input is via the push button methods. Any object capable of being rotated can act as a selector switch (e.g. values, dampers, etc.).

6.5.3.6.1 MB_2PosSSDisplay(svgdoc, SSID, offangle, onangle)

Two Position Selector Switch Control. This rotates an element to one of two angles, depending on whether the monitored value is off ("0") or on (non-zero).

- **svgdoc** - Reference to SVG document.
- **SSID (string)** - ID of SVG item.
- **offangle (integer)** - Angle in degrees when value is off (0).
- **onangle (integer)** - Angle in degrees when value is on (non-zero).

```
// Pick and place selector switch control for vertical axis.  
var PPSSVertDisplay = new MB_2PosSSDisplay(document, "SSPPVert", -60,  
60);  
MBHMIProtocol.AddToDisplayList(PPSSVertDisplay, "SSPPVert", "read");
```

6.5.3.6.2 MB_3PosSSDisplay(svgdoc, SSID, negativeangle, zeroangle, positiveangle)

Three Position Selector Switch Display. This rotates an element to one of three angles, depending on the sign (-ve, +ve, or 0) of the monitored value.

- **svgdoc** - Reference to SVG document.
- **SSID (string)** - ID of SVG item.
- **negativeangle (integer)** - Angle in degrees when value is negative.
- **zeroangle (integer)** - Angle in degrees when value is zero.
- **positiveangle (integer)** - Angle in degrees when value is positive.

```
// Pump control selector switch position.  
var PumpSSSwitchDisplay = new MB_3PosSSDisplay(document, "PumpSSwitch",  
-60, 0, 60);
```



```
MBHMIProtocol.AddToDisplayList(PumpSSwitchDisplay, "PumpSpeedActual",
"read");
```

6.5.3.7 Numeric and Text Displays

Numeric and text displays display numbers and text.

6.5.3.7.1 MB_NumericDisplay(svgdoc, NumberID)

Update a display of a number. This operates by changing the text property of an SVG string.

- `svgdoc` - Reference to SVG document.
- `NumberID (string)` - ID of SVG text item.

```
// Pump speed numeric display.
var PumpSpeedDisplay = new MB_NumericDisplay(document,
"PumpSpeedText");
```

6.5.3.7.2 MB_NumericFloatDisplay(svgdoc, NumberID, Decimals)

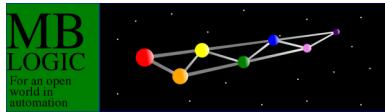
Update a display of a floating point number with control of the number of decimals displayed. This operates by changing the text property of an SVG string.

- `svgdoc` - Reference to SVG document.
- `NumberID (string)` - ID of SVG text item.
- `Decimals (integer)` - Number of decimal places.

```
// Tank 1 fill level numeric display.
var Tank1Number = new MB_NumericFloatDisplay(document, "Tank1Text",
2);
MBHMIProtocol.AddToDisplayList(Tank1Number, "Tank1Number", "read");
```

6.5.3.7.3 MB_StringDisplay(svgdoc, NumberID)

Update a display of a string. This operates by changing the text property of an SVG string.



- `svgdoc` - Reference to SVG document.
- `NumberID (string)` - ID of SVG text item.

```
// Text string display.  
var DemoStringDisplay = new MB_StringDisplay(document, "DemoString");
```

6.5.3.7.4 MB_TextDisplay(svgdoc, TextMsgID, MsgList)

Update a display of a text message. This operates by changing the text property of an SVG string. The text messages must be defined in an array of strings. This array determines what text message is displayed for each integer value.

- `svgdoc` - Reference to SVG document.
- `TextMsgID (string)` - ID of SVG text item.
- `MsgList (array)` - Array of strings containing messages.

```
// This is for text indicators.  
Msg_HMIDemoColours = ["red", "orange", "yellow", "green", "blue",  
"indigo", "violet"]  
// This is for the text display of pilot light colours.  
var PLColourText = new MB_TextDisplay(document, "PL4-Colour",  
Msg_HMIDemoColours);
```

6.5.3.8 Gauges, Columns, Pipes, and Pumps

Gauges, columns, pipes and pumps operate on various attributes, including fill and rotate. These functions can operate on any SVG graphic which has the appropriate attribute. This includes conveyors, ducts, etc.

6.5.3.8.1 MB_DialGauge(svgdoc, DialID, DialMin, DialMax, DataMin, DataMax)

Dial Gauge Indicator Control. This changes the rotate property of an SVG element. All angles are in degrees. The dial display angle is calculated as follows: (((newdata - DataMin) / DataMax) + DialMin) * (DialMax - DialMin)

- `svgdoc` - Reference to SVG document.
- `DialID (string)` - ID of SVG item.
- `DialMin (integer)` - The minimum dial angle to use.



- DialMax (integer) - The maximum dial angle to use.
- DataMin (integer) - The minimum data range.
- DataMax (integer) - The maximum data range.

On start up, the function will search for a "g" SVG tag within the SVG widget and apply the rotation to that. This means that any geometric shape can be rotated, but it must be enclosed within a group (<g></g>).

```
// Dial gauge graphic display. This is tied to Tank 1 level
var Dial1 = new MB_DialGauge(document, "Dial1", 30, 330, 0, 4095);
```

6.5.3.8.2 MB_ColumnGauge(svgdoc, GaugeID, GradientID, MinData, MaxData)

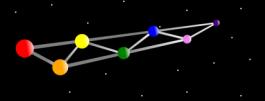
Column level (including tank cut-outs) control. This operates by adjusting the linear gradient that is used to fill a "column" (any closed SVG object). A total of four "gradient stops" are used. The first and last are fixed at 0% and 100% respectively. The second and third are set to the value in the update parameter (which must be between 0 and 100%). By setting two adjacent gradient stops to the same value, a sharp colour transition is created, instead of a gradual one. The data is automatically scaled to 0% to 100% based on the MinData and MaxData parameters.

- svgdoc - Reference to SVG document.
- GaugeID (string) - ID of the SVG item.
- GradientID (string) - ID of the linear colour gradient prototype to be used.
- MinData (integer) - The minimum data range.
- MaxData (integer) - The maximum data range.

On start up, the function will search for a geometric shape to apply the fill to. The order of search is as follows:

1. polygon
2. rect
3. circle
4. ellipse

It also looks for the specified colour gradient (from the **GradientID** parameter) and makes a new copy of it. It then gives it an new id consisting of the column id (from



the **GaugeID** parameter), plus "_FillGradient". It also creates ids for the gradient stops also based on the column id, plus "_StopA" and "_StopB". This results in a new gradient used by this column only.

```
// Tank 2 fill level graphic display.
var Tank2 = new MB_ColumnGauge(document, "Tank2ID",
"MB_Column_BlueGradient", 0, 500);
```

6.5.3.8.3 MB_Pipe2(svgdoc, PipeID, initcolour, offcolour, oncolour)

Indicate conditions in pipes or ducts. This is similar to MB_Pipe, but operates by adjusting the fill colour instead of the stroke.

- **svgdoc** - Reference to SVG document.
- **PipeID** (string) - ID of SVG item.
- **initcolour** (string) - Colour to use when undefined.
- **offcolour** (string) - Colour to use when zero.
- **oncolour** (string) - Colour to use when not zero.

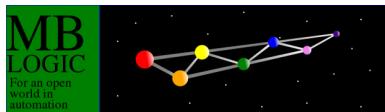
```
// Pipe flow using pipes which change colour instead of animating.
// This is different from a pilot light in that any non-zero value
// is considered to be 'on'.
var Tank2Pipe = new MB_Pipe2(document, "Tank2Pipe", "black", "green",
"red");
```

6.5.3.8.4 MB_PumpRotate(svgdoc, PumpID, rotation)

Pump or fan control. This operates by adjusting the rotate property of the SVG element. Each update, the rotation amount is incremented by the amount specified in the "rotation" parameter.

- **svgdoc** - Reference to SVG document.
- **PumpID** (string) - ID of pump SVG item.
- **rotation** (integer) - Number of degrees to rotate each time. From -360 to 360.
+ve = cw. -ve = ccw.

```
// Pump rotation.
var PumpRotation = new MB_PumpRotate(document, "Pump1", 19);
```



6.5.3.8.5 MB_PumpRotateAnimated(svgdoc, PumpID)

Animated pump rotation control. This provides a smooth continuous animation that does not depend on the scan rate. The rotation is on or off, and in one direction only. This uses the SVG animateTransform property.

- `svgdoc` - Reference to SVG document.
- `PLID (string)` - ID of SVG item.

```
// Animated pump.  
var PumpAnil = new MB_PumpRotateAnimated(document, "PumpAnil");  
MBHMIProtocol.AddToDisplayList(PumpAnil, "PumpAnil", "read");
```

6.5.3.9 Miscellaneous

6.5.3.9.1 MB_SlideDisplay(svgdoc, SlideID, offposx, offposy, onposx, onposy)

Two Position Slide Control. This translates (slides) an element to one of two positions, depending on whether the monitored value is off ("0") or on (non-zero).

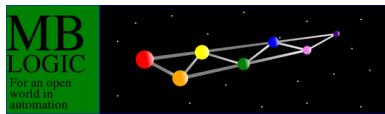
- `svgdoc` - Reference to SVG document.
- `SSID (string)` - ID of SVG item.
- `offposx, offposy (integer)` - X & Y positions in pixels when value is off (0).
- `onposx, onposy (integer)` - X & Y position in pixels when value is on (non-zero).

```
// Pick and place vertical axis.  
var PPVertDisplay = new MB_SlideDisplay(document, "PPVert", 0, 0, 0,  
100);  
MBHMIProtocol.AddToDisplayList(PPVertDisplay, "PPVert", "read");
```

6.5.3.9.2 MB_GraphicSelect(pagedoc, graphictable, tableoffset)

Select an SVG image from a list (array) of images. Each image must have a unique "id" string. This function hides or displays different parts to display only one at a time. This is compatible with the display list.

- `pagedoc` - A reference to the HTML document.



- **graphictable** (array) - An array of strings containing the ids of the SVG graphic images.
- **tableoffset** (integer) - This value is added to the monitored value in determining the index for the "graphictable". This allows data ranges which do not start at zero to be used directly as image selectors (e.g. such as 3 position selector switches).

```
// Select a shape for display.  
var ShapeDemoDisplay = new MB_GraphicSelect(document,  
    ["ShapeDemoCir", "ShapeDemoRect", "ShapeDemoOct"], 1);  
MBHMIProtocol.AddToDisplayList(ShapeDemoDisplay, "ShapeDemo", "read");
```

6.5.3.9.3 MB_GraphicShowHide(svgdoc, GraphicId)

Display or hide an SVG image. 0 = hide the image. 1 = Show the image. A typical application is to use it to control an enable/disable "mask" over a push button. Hiding the mask will "enable" the push button (uncover it, while showing the mask will "disable" the push button (cover it so it cannot be activated). This is compatible with the display list.

- **svgdoc** - Reference to SVG document.
- **PLID (string)** - ID of SVG item.

```
var PB1Mask = new MB_GraphicShowHide(document, "PB1Mask");  
MBHMIProtocol.AddToDisplayList(PB1Mask, "PB1Mask", "read");
```

6.5.3.10 Strip Charts

6.5.3.10.1 MB_StripChart(svgdoc, ChartID, MaxPoints, XInc, YScale, TimeInterval)

Update a strip chart. This operates by changing the points property of an SVG polyline.

- **svgdoc** - Reference to SVG document.
- **ChartID (string)** - ID of SVG polyline.
- **MaxPoints (integer)** - Maximum number of chart points.
- **XInc (integer)** - Increment for each X position.



- **YScale** (integer) - Scale factor to apply to readings.
- **TimeInc** (float) - Minimum time increment in seconds.

```
// Strip charts.  
var DemoChart1 = new MB_StripChart(document, "demochart1", 50, 10, 1,  
1.0);  
MBHMIProtocol.AddToDisplayList(DemoChart1, "StripChart1", "read");
```

6.5.3.11 Data Entry

Data entry is also possible using normal push button increment/decrement operations. See the section on push buttons for more information.

6.5.3.11.1 MB_NumericPad(svgdoc, NumberID, MaxDigits)

Implement a numeric entry pad. This does not use the display list.

- **svgdoc** - Reference to SVG document.
- **NumberID** (string) - ID of SVG text item.
- **MaxDigits** (integer) - Maximum number of digits permitted.

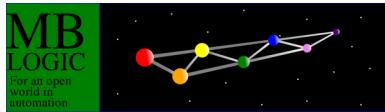
```
// Numeric display. This does not get added to the display list.  
var MBT_NumberPad = new MB_NumericPad(document,  
"MBT_NumberPaddir", 12);
```

The numeric pad expects the Javascript object to be named "MBT_NumberPad". You use a separate numeric "STR" (store) button to write the accumulated value of the numeric pad to the server. You do *not* add the "MBT_NumberPad" object to the display list.

6.5.3.12 Screen Select Control

6.5.3.12.1 MB_ScreenSelect(pagedoc, screentable)

Select which HMI "screens" are visible. The "screens" are all parts of the same xhtml document. This function hides or displays different parts to give the illusion of



different screens being selected. This operates by changing the "display" property of the style between "none" and "block". When activated, all screens in the "screentable" list are set to "none", then the selected screen is set to "block".

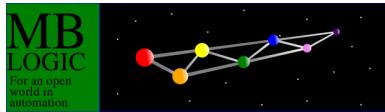
MB_ScreenSelect does not work with the display list. It is operated directly by calling the SelectScreen property with a string parameter containing the id of the desired screen.

- pagedoc - A reference to the HTML document.
- screentable (array) - An array of strings containing the ids of the screen divs.

The following example shows how MB_ScreenSelect interacts with "onclick" to create HTML menus. If you use the provided page templates these menus are already created for you. See the documentation on adding menu buttons to select SVG graphical screens to see how to use this with drag and drop editing using Inkscape.

```
// Make a list of all the screens that can be selected.
var ScreenTable = ["mainscreen", "eventscreen", "alarmscreen"];
// This creates an object that controls display of the screens.
var ScreenSelect = new MB_ScreenSelect(document, ScreenTable);

<!-- This creates the HTML menu. -->
<div id="nav">
<ul>
<li><a onclick = "ScreenSelect.SelectScreen('mainscreen')">
Main</a></li>
<li><a onclick = "ScreenSelect.SelectScreen('eventscreen')">
Events</a></li>
<li><a onclick = "ScreenSelect.SelectScreen('alarmscreen')">
Alarms</a></li>
</ul>
</div>
```



6.5.4 HMI Client Event Library

6.5.4.1 Overview

The HMI system is based on open standards such as JSON, Javascript, SVG, and HTML. The HMI client event library is not required to build a web based HMI which can interact with the HMI server. However, the HMI client events library provides a number of useful functions which make creating web based HMI systems easier. These functions are documented below.

The client event library ("*libmbevents.js*") is implemented as a series of independent objects. Most of these objects are designed to be compatible with the HMI communications library "display list".

6.5.4.2 Event Display

"Events" is the standard term for occurrences which an operator must be notified of, but which does not require acknowledgement.

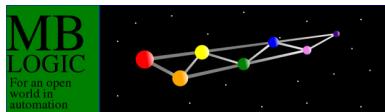
6.5.4.2.1 Parameters

- **eventdoc** - A reference to the HTML document.
- **eventtableID (string)** - The id of the html table used to display events.
- **maxlength (integer)** - The maximum number of rows to display.
- **eventtexts (array)** - An array of strings with the event messages.

```
function MB_EventDisplay(eventdoc, eventtableID, maxlength,  
eventtexts)
```

MB_EventDisplay requires one table with four columns. For example -

```
<table id="EventDisplay" border="5" cellpadding="5">  
<tr>  
<td><b>Event #:</b></td>  
<td><b>Date:</b></td>  
<td><b>Event:</b></td>  
<td><b>State:</b></td>
```



```
</tr>
</table>
```

The first column will be the Event number. The second column is the date, the third column is the event name, and the fourth column is the event state. These column assignments are fixed, and cannot be changed without editing the code.

The Javascript code to display events is shown below.

```
// This is to display the events.
var EventDisplay = new MB_EventDisplay(document, "EventDisplay", 50,
event_text);
    // Add this to the display list.
    MBHMIProtocol.AddToDisplayList(EventDisplay, "events", "events");
```

The texts displayed on the screen are not carried in the actual messages, but must be provided by the client. The messages are expected to be in object literals with the message tags acting as keys to the actual messages. An easy way of implementing this is to put the messages into external Javascript files containing just the data definitions. For example:

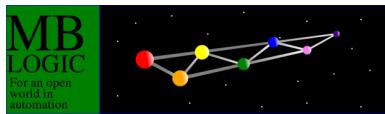
```
event_text = {
    "PumpRunning" : "Tank pump is running.",
    "PumpStopped" : "Tank pump is stopped.",
    "Tank1Empty" : "Tank 1 is empty.",
    "Tank1Full" : "Tank 1 is full.",
    "Tank2Empty" : "Tank 2 is empty.",
    "Tank2Full" : "Tank 2 is full."
}
```

6.5.4.3 Alarms Display

"Alarms" is the standard term for occurrences which an operator must be notified of, and which *must* be acknowledged.

6.5.4.3.1 Parameters

- `alarmdoc` - A reference to the HTML document.
- `alarmtableID (string)` - The id of the html table used to display alarms.



- **alarmtexts** (object) - An object of strings with the alarm messages.
- **alarmstatetexts** (object) - An object of strings with the alarm states.
- **alarmcolour** (string) - The HML colour to indicate alarm conditions.
- **ackcolour** (string) - The HML colour to indicate acknowledged conditions.
- **okcolour** (string) - The HML colour to indicate OK conditions.

```
function MB_AlarmDisplay(alarmdoc, alarmtableID, alarmtexts,  
alarmstatetexts,  
                        alarmcolour, ackcolour, okcolour)
```

MB_AlarmDisplay requires one table with five columns. For example -

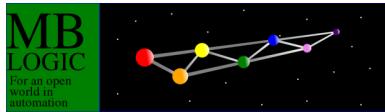
```
<table id="AlarmDisplay" border="5" cellpadding="5">  
<tr>  
<td><b>Alarm:</b></td>  
<td><b>Alarm State:</b></td>  
<td><b>Time:</b></td>  
<td><b>Time OK:</b></td>  
<td><b>Count:</b></td>  
</tr>  
</table>
```

The first column will display the name of the alarm. The second column will display the alarm state. The third column will display the time at which the fault was detected. The fourth column will display the time at which the fault became OK. The fifth column will display the number of times the fault occurred while the alarm was displayed.

The Javascript code to display alarms is shown below.

```
// This is to display the alarms.  
var AlarmDisplay = new MB_AlarmDisplay(document, "AlarmDisplay",  
                                         alarm_text, alarmstates_text, "red", "orange", "green");  
// Add this to the display list.  
MBHMIProtocol.AddToDisplayList(AlarmDisplay, "alarms", "alarms");
```

The texts displayed on the screen are not carried in the actual messages, but must be provided by the client. The messages are expected to be in object literals with the message tags acting as keys to the actual messages. An easy way of implementing this



is to put the messages into external Javascript files containing just the data definitions. For example:

```
alarm_text = {
    "PB1Alarm" : "PB1 was pressed.",
    "PB2Alarm" : "PB2 was pressed.",
    "PB3Alarm" : "PB3 was pressed.",
    "PB4Alarm" : "PB4 was pressed."
}
```

The texts used to describe the alarm states are not carried in the actual messages, but must be provided by the client. The messages are expected to be in object literals with the message tags acting as keys to the actual messages. An easy way of implementing this is to put the messages into external Javascript files containing just the data definitions. For example:

```
alarmstates_text = {
    "alarm" : "Fault is active",
    "ackalarm" : "Alarm acknowledged",
    "ok" : "Fault cleared",
    "ackok" : "Fault cleared and acknowledged",
    "inactive" : "Alarm inactive"
}
```

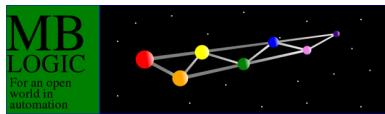
MB_AlarmDisplay does not provide a function to input the alarm acknowledge action. However an appropriate function is provided in the libhmiclient library which can be attached to a button on the screen.

```
<!-- Push button to acknowledge alarms. -->
<g transform="translate(50, 10)"
onclick="MBHMIProtocol.AddAlarmAck();">

    <!-- Put some SVG or other suitable mark up in here to display
a button -->

</g>
```

6.5.4.4 Alarms History Display



"Alarms" is the standard term for occurrences which an operator must be notified of and acknowledge. Alarm history is a record of past alarms.

6.5.4.4.1 Parameters

- alarmhistorydoc - A reference to the HTML document.
- alarmhistorytableID (string) - The id of the html table used to display alarm history.
- maxlenlength (integer) - The maximum number of rows to display.
- alarmtexts (array) - An array of strings with the alarm messages.

```
function MB_AlarmHistoryDisplay(alarmhistorydoc, alarmhistorytableID,
maxlength, alarmtexts)
```

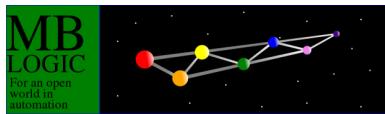
MB_AlarmHistoryDisplay requires one table with four columns. For example -

```
<table id="AlarmHistoryDisplay" border="5" cellpadding="5">
<tr>
<td><b>Alarm:</b></td>
<td><b>Alarm Time:</b></td>
<td><b>Time OK:</b></td>
<td><b>Ack By:</b></td>
</tr>
</table>
```

The first column is the alarm name. The second column will display the time at which the fault was detected. The third column will display the time at which the fault became OK. The fourth column will display who (which client ID) acknowledged the alarm.

The Javascript code to display alarm history is shown below.

```
// This is to display the alarm history.
var AlarmHistoryDisplay = new MB_AlarmHistoryDisplay(document,
    "AlarmHistoryDisplay", 50, alarm_text);
// Add this to the display list.
MBHMIProtocol.AddToDisplayList(AlarmHistoryDisplay, "alarmhistory",
"alarmhistory");
```



The texts displayed on the screen are not carried in the actual messages, but must be provided by the client. The messages are expected to be in object literals with the message tags acting as keys to the actual messages. An easy way of implementing this is to put the messages into external Javascript files containing just the data definitions. For example:

```
alarm_text = {
  "PB1Alarm" : "PB1 was pressed.",
  "PB2Alarm" : "PB2 was pressed.",
  "PB3Alarm" : "PB3 was pressed.",
  "PB4Alarm" : "PB4 was pressed."
}
```

6.5.4.5 Error and Status Display

Error and Status display shows protocol errors and communications status in HTML tables. These require that certain HTML tables already exist for them to be associated with. They then add and remove rows of data to or from these tables to display logs of communications errors.

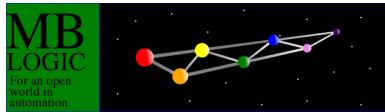
Errors and communications status are not buffered in the server. Reloading the browser will cause the existing record to be erased.

MB_TagErrorDisplay requires one table with three columns. The first column will be the current date and time. The second column is the name of the tag which encountered an error, and the third column is the text description of the error. These column assignments are fixed, and cannot be changed without editing the source code.

6.5.4.5.1 MB_TagErrorDisplay(errordoc, errortableID, maxlen, errortexts)

Display protocol errors on an HTML screen. Parameters:

- **errordoc** - A reference to the HTML document.
- **errortableID (string)** - The id of the html table used to display errors.
- **maxlength (integer)** - The maximum number of rows to display.
- **errortexts (array)** - An object with the error messages.



```
// This is to display the communications errors.  
var ErrorDisplay = new MB_TagErrorDisplay(document, "ErrorDisplay",  
50, error_text);  
    // Add this to the display list.  
    MBHMIProtocol.AddToDisplayList(ErrorDisplay, "errors", "errors");
```

MB_StatusLogDisplay requires one table with three columns. The first column will be the current date and time. The second column is the name of the status code, and the third column is the text description of the status code. These column assignments are fixed, and cannot be changed without editing the source code.

6.5.4.5.2 **MB_StatusLogDisplay(statusdoc, statustableID, maxlenlength, statustexts)**

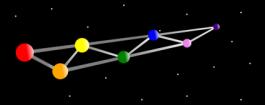
Display protocol status conditions on an HTML screen. Parameters:

- **statusdoc** - A reference to the HTML document.
- **statustableID (string)** - The id of the html table used to display status.
- **maxlength (integer)** - The maximum number of rows to display.
- **statustexts (array)** - An object with the status messages.

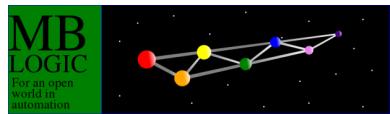
```
// This is to display the communications status log.  
var StatusLogDisplay = new MB_StatusLogDisplay(document,  
"StatusDisplay", 50, status_text);  
    // Add this to the display list.  
    MBHMIProtocol.AddToDisplayList(StatusLogDisplay, "status", "status");
```

The text for the descriptions are not carried in the actual messages, but must be provided by the client. The text of the descriptions are expected to be in object literals with the message tags acting as keys to the actual messages. An easy way of implementing this is to put the messages into external Javascript files containing just the data definitions. For example:

```
error_text = {"tagnotfound" : "The address tag is not recognised by  
the server.",  
            "badtype" : "The data value is of an incompatible type.",  
            "outofrange" : "The data value is out of range.",  
            "writeprotected" : "An attempt was made to write to an address which  
is write  
                protected or otherwise not writable.",  
            "addresserror" : "An error occurred in attempting to map the tag to  
the
```



```
internal server address representation.",  
    "servererror" : "An unspecified error has occurred in the server which  
prevents  
        the request from being completed.",  
    "accessdenied" : "The client does not have authorisation to access  
this tag."  
}  
  
status_text = {  
    "ok" : "No errors.",  
    "protocolerror" : "An error was encountered in the protocol and the  
entire message  
        was discarded by the server.",  
    "commanderror" : "A request command field provided incorrect or  
invalid data.",  
    "servererror" : "An unspecified error has occurred in the server which  
prevents  
        the request from being completed.",  
    "unauthorised" : "The client is not authorised to communicate with  
this server.",  
    "noclistempty" : "The client attempted to read using NOC without an  
NOC list being  
        present in the server."  
}
```



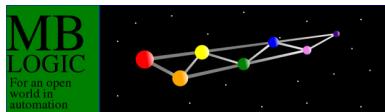
6.5.5 Basic Concepts

6.5.5.1 Overview

This section describes the basic concepts behind the operation of a web based HMI. This section is of interest to those interested in understanding the underlying principles of operation, and to those who wish to build their own custom systems without using any of the provided templates.

6.5.5.1.1 Concepts

- [HMI Web Basics](#) - Introduction to basic web page concepts.
- [HMI Server Functions](#) - Introduction to HMI server features.
- [Building Web Clients](#) - Making a web based HMI Client.
- [Javascript](#) - Overview of Javascript in the HMI.
- [HMI SVG Graphic Elements](#) - Using the SVG graphics samples, and creating your own graphics.



6.5.5.2 HMI Basics

6.5.5.2.1 Overview

The HMI subsystem is a Human-Machine Interfaced (HMI) based on standard web technologies. This allows both local and remote HMI systems to be created for machine interaction using standard web browsers.

6.5.5.2.2 Standard Web Technologies

The HMI is based on the following web technologies:

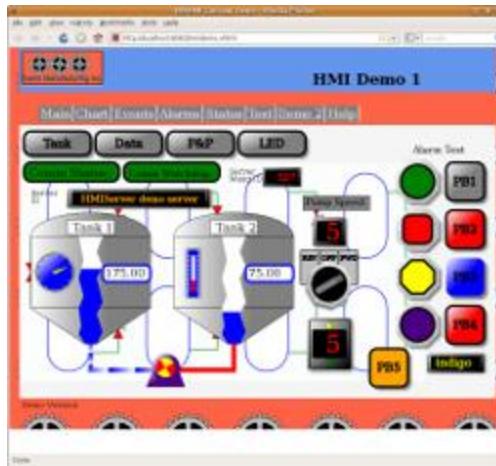
- HTML (Hyper Text Mark-up Language). HTML is the basis of virtually every web page, and is used as the foundation for the system's HMI screens.
 - Javascript. Javascript (not to be confused with Java) is the common scripting language for web pages, and is used by the HMI to control the display of data and to exchange data with the server.
 - SVG (Scalable Vector Graphics) - SVG allows high quality interactive graphics to be embedded directly into web pages, and is supported by almost every modern web browser. SVG is used by the HMI to provide the graphical elements to displays.
 - CSS, JPEG, PNG, JSON - Since the HMI is based on standard web technologies, it can take advantage of standard web styling and graphics.
 - HMI Protocol - A JSON (Javascript Object Notation) based open communications protocol. This protocol was developed to provide a simple and reliable means of communications with low overhead and high level functionality.
-

6.5.5.2.3 HMI Components

The HMI includes the following components:

- A standard xHTML web page which can be used as the starting point for new HMI designs.

- A standard CSS style sheet which can be modified to customise the look and feel of a new application. CSS (Cascading Style Sheet) is one of the foundations of modern web pages. This allows the definitions of things like colours, fonts, spacing, and other characteristics to be defined separately from the actual page content, making pages more manageable and maintainable. It also allows for special effects in menus and graphics.
- A Javascript library to provide communications between the browser and the server using the HMI protocol. It also stores the data in a common data table, provides means of accessing this data, and automatically updates the screen display elements when new data arrives.
- A Javascript library providing control and animation of display elements, including SVG graphics elements, and standard event and alarm display.
- A selection of SVG graphic elements that can be used in new applications, or used as examples for developing custom graphics.



6.5.5.2.4 Browser Compatibility

Since the HMI is based on web standards, it is compatible with most modern web browsers. The following web browsers have been tested:

- Firefox on Linux
- Epiphany on Linux
- Opera on Linux
- Midori on Linux
- Google Chromium on Linux

- Firefox on MS Windows
- Opera on MS Windows
- Google Chrome on MS Windows
- Apple Safari on MS Windows

Testing has not been conducted on the Apple MacIntosh, but there is no reason to believe it will not work equally well there. The MS Internet Explorer web browser does not work, as it is based on outdated technologies and does not support the features needed to provide a useful graphical HMI.

6.5.5.2.5 Web Page Basics

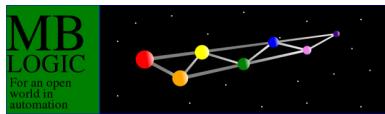
Web pages are made from a combination of text, mark-up (HTML) symbols, Javascript (a programming language), and images. What gives them structure and style is the HTML mark-up (formatting).

6.5.5.2.6 HTML versus XHTML

SVG graphics can be directly incorporated into web pages. The most commonly used version of ordinary HTML (version 4) does not at present support SVG "in-line", that is, directly in the page. SVG images can be loaded from external files, but this is not convenient for most industrial HMI applications. The upcoming replacement version (version 5) will support SVG "in-line", but it is not yet available in most web browsers yet.

However, there is a standard variation on HTML called "XHTML" which does allow in-line SVG. The differences between HTML and XHTML are relatively minor, the main one being that XHTML tends to be much more strict about formatting. Anyone familiar with creating a standard HTML 4 web page should have no difficulties when dealing with XHTML.

6.5.5.2.7 Creating Web Pages



Web pages consist of simple text and formatting codes (called "mark-up"). These mark-up codes are detailed in numerous books and web sites and will not be discussed in detail here.

Unlike word processor documents, web pages are not normally constructed using WYSIWIG editors. They are usually created using text editors. The reason for this is that WYSIWIG editors usually produce very poor quality output which often fails in unexpected ways. Furthermore, most web sites today are "dynamic", which means that each web page is actually created by a program when you request to see it. Creating a web page has become more like writing a program than like typing a document. The combination of these means that there is little demand for WYSIWIG editors for web pages.

However, there are many editors which "understand" HTML syntax and will highlight and even complete the mark-up codes as you enter them. A good HTML editor won't create a web page for you, but it will help you avoid errors. Like most development software today, many of the best HTML editors are free and can be downloaded from many different places. These web pages were created using an editor called "Bluefish".

6.5.5.2.8 CSS

"CSS" stands for "Cascading Style Sheets". CSS is used to describe the appearance of a web page separately from the content. This allows web sites to adopt a common look and feel while still being manageable. The CSS codes are normally kept in a separate file (with the extension ".css") and imported into the web pages.

CSS is also important to HMI applications because it allows special effects to be used which give a better looking and more usable HMI. The details of this will be discussed in the section on constructing a complete HMI application.

6.5.5.2.9 Raster Image Formats



A "raster" image format is sometimes also called a "bit-map" image. Raster images are often used for backgrounds, logos, and "favicons" (small icons often appearing beside the browser address bar and in the bookmarks menu). The two common formats for images are "PNG" and "JPEG". JPEG images are best used for photographs. For drawn graphics, PNG has largely replaced the older GIF image format. For drawn graphics, PNG will usually give a better looking graphic than JPEG.

In industrial HMI applications, JPEG and PNG images can be used to display things like company logos, small drawings or photographs of the equipment, drawings or photographs of the product being manufactured, and other relevant information. These can be used to provide an attractive and informative user interface.

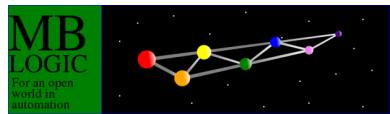
6.5.5.2.10 Javascript

Javascript is a programming language which is supported by almost all modern web browsers. Despite the name, Javascript has no relationship to the Java programming language. They are two different languages, which both happen to have a syntax that was largely inspired by 'C'.

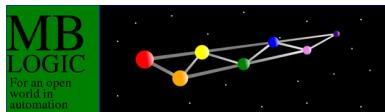
Javascript belongs to what are sometimes referred to as "scripting" languages. A scripting language is a language which is intended for writing applications, as opposed to "systems" languages, which are intended for writing things like operating systems or compilers. Like most scripting languages, Javascript has many powerful features based on object oriented programming concepts. The details of Javascript programming are not discussed here. There are many books and web sites devoted to Javascript programming which should be consulted for that information.

6.5.5.2.11 SVG

SVG stands for Scalable Vector Graphics. SVG is one of the web standards specified by the W3C, which is the organisation responsible for defining and documenting the standards used by modern web browsers. SVG allows high quality interactive graphics to be embedded directly into a web page.



Unlike simple PNG or JPEG graphics, SVG is an XML based description of a graphic element which can be directly manipulated by client scripting. That is, the SVG graphic or image can be changed dynamically without having to reload the web page or download a new image. This allows graphical web based HMI applications to be created which have the sort of interactive graphics normally associated with conventional HMI programs.



6.5.5.3 HMI Server Functions

6.5.5.3.1 Overview

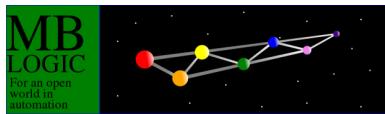
A web based client cannot operate on its own. It requires a server to load the client web page from, and to communicate with. This system has a special server built into it which provides both functions.

6.5.5.3.2 Server Features:

The HMI server provides the following features to an application.

- A web server. The server will load web pages requested by a web browser. This includes the web page itself, external Javascript files, CSS files, graphics images such as PNG or JPEG files, and icons ("favicons").
 - A common data table. All server data is stored in the common system (communications) data table. This means that a client can interact with other protocols and devices through the common data table. This includes monitoring inputs, setting outputs, and viewing events and alarms. (***This is in MBLogic only. In HMIServer data is read directly from the external source.***)
 - Events and alarms. The server detects events and alarms, time stamps them, constructs the messages, stores them in an event queue where they can be requested by the client, and responds to alarm acknowledge messages. More details on events and alarms can be found below.
 - Read only tags. Some tags can be defined as "read only" to protect their contents from being inadvertently changed.
 - Range limits and scaling. Numeric tags can have range limits and scale factors applied by the server. Range limits can be used to prevent invalid data from being accepted by the server. Scale factors allow server data table values to be converted to a different range (e.g. convert to engineering units) for the client.
 - A configuration system. Tags, data table addresses, and other features are all configurable. More details on configuration can be found below.
-

6.5.5.3.3 Events and Alarms



Events and alarms are represented as boolean conditions. An event occurs when the boolean value changes from false to true, or from true to false. An alarm occurs when the boolean value changes from false to true.

The event and alarm manager does not directly "create" events or alarms by monitoring conditions such as analogue values. Events and alarms must be created by other logic (e.g. via the soft logic or by other systems).

The addresses used for events and alarms can be anywhere in the coils area of the data table. External protocol clients, internal clients and integrated software modules can all write to the coils area to trigger an event or alarm.

6.5.5.3.4 Event and Alarm Buffering

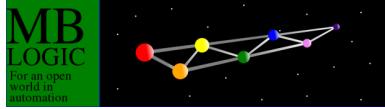
Events and alarms are "buffered" in the server. This means that when an event or alarm occurs, a copy of each message is retained in the server so that a client can retrieve all or some of them later. This means the client does not need persistent (disk based) local storage to maintain a (recent) history of events and alarms. They can be retrieved from the server at any time.

At present, the system stores the event and alarm buffers in memory. This means they do not (at present) retain a permanent event or alarm history. If the server is restarted, the existing history is lost. Permanent event and alarm storage will be addressed in a future version.

6.5.5.3.5 Data Scaling and Conversion

The HMI server offers automatic scaling of numbers as part of the configuration system. Values in the data table may be automatically scaled when read from the server, with the reverse scaling being applied on writing data to the server. Scaling is user selected parameters as part of the configuration.

The server also automatically converts data types as required. Conversion is implemented as follows:



6.5.5.3.5.1 Scaling When Receiving Data

- On receiving a value which is intended to be written to a register, the server first attempts to convert it to an integer. If this fails, it attempts to convert it to a floating point number. If that in turn fails, it reports it as a 'badtype' error.
- The server then checks to see if the number is within the user defined range. If not, then it reports it as an 'outofrange' error.
- Next, the server attempts to apply the scale factors. The equation used is $\text{scaledvalue} = (\text{datavalue} - \text{scaleoffset})/\text{scalespan}$. If an error occurs, it is reported as a 'servererror'.
- Next, the scaled value is checked to see if it will fit within a data table register.
- If there are no failures at this point, the value is converted to an integer and stored in the data table.

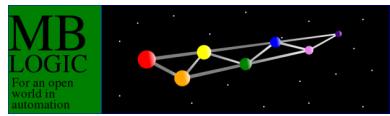
6.5.5.3.5.2 Scaling When Sending Data

- When sending data, the value is first read from the data table.
- Next, the server attempts to apply the scale factors. The equation used is $\text{scaledvalue} = (\text{scalespan} * \text{datavalue}) + \text{scaleoffset}$. If an error occurs, it is reported as a 'servererror'.
- The scaled value is next converted to the configured data type. If an error occurs, it is reported as a 'badtype'.
- Finally, the the number is checked to see if it is within the user defined range. If not, then it reports it as an 'outofrange' error.

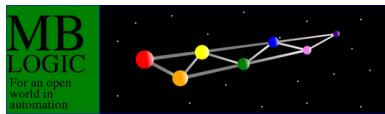
6.5.5.3.6 Platform Data Limits

Because all data values are mapped to data table addresses, the data must exist within the limits of that data table. These are limitations of the server platform and not HMI protocol limits.

- Native integers must be within the range -32768 to 32767 (after scaling is applied).
- Extended data types offer integers in the range -2147483648 to 2147483647. However, these require two registers per value.
- Native floating point numbers are automatically converted to integers when stored in the data table.



- Extended data types offer single and double precision floating point. However, these require 2 (single precision) or 4 (double precision) registers per value.
- There are no limits on string length, but a single register may hold only 1 or 2 characters.
- Discrete inputs and input registers are read-only addresses.



6.5.5.4 HMI Web Clients

6.5.5.4.1 Overview

This web based HMI system is an open system based on web standards. Therefore there are no hard and fast rules as to how an HMI may be built. It is possible to build a web based HMI using the HMI protocol using your own components and without using any of the components described here. However, the software components provided with the system make building an HMI faster and easier. This section concentrates on building a simple example.

6.5.5.4.2 Web Based Applications

There are two major types of web based applications. In the traditional type, the control logic is all performed on the server, with the client (web browser) being limited to just displaying static HTML from the server. With this sort of application, updating any of the information on the client requires loading a new web page from the server. This can be a comparatively slow process which is not suited to displaying rapidly changing information.

A newer type of web application, often referred to as "AJAX" (Asynchronous Javascript And XML), has been developed however which solves this shortcoming. This uses the ability of the web browser to fetch new information from a server without reloading the entire page. The result is the user will see the information on the page update on a continuous basis without being interrupted for a page reload.

There are several means of accomplishing this reading of information. The one most suited to industrial HMI applications is a Javascript function called "XMLHttpRequest". Despite the name, XMLHttpRequest can be used to fetch any sort of data, not just XML. In this HMI application, the protocol uses it to fetch JSON encoded data.

6.5.5.4.3 Security and 'Same Origin'



A web based HMI normally involves a web page and a web browser. The web browser will request the web page from a web server using what is referred to as an http "GET" operation. An AJAX application can then use XMLHttpRequest to request updated information from the server using what are referred to as http "POST" operations ("GET" can also be used in some circumstances).

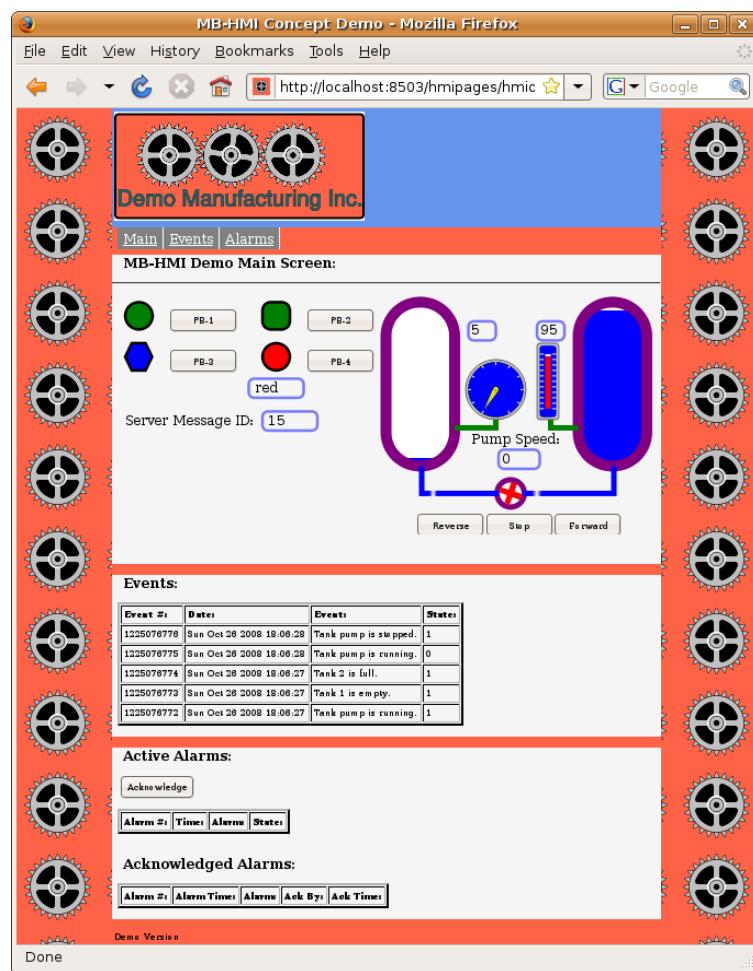
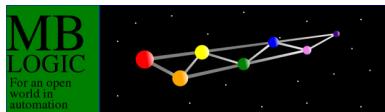
Most web browsers however have a limitation which they impose for security reasons. The limitation is that an AJAX client can only communicate with the same server that the web page came from. This means it is not possible to fetch web pages from one server (or open them from a local disk) and request data from a different server. It also means that it is not possible for an HMI application to fetch data from multiple sources simultaneously.

This is not normally a serious limitation for most applications. Most HMI applications will only need to interact with one server. If they do need information from several servers, then it is usually possible for one server to fetch this information so the HMI client can read it from a single source.

6.5.5.4.4 Web Applications and Web Pages

The first step in building a web based HMI is creating a web page. A typical HMI will consist of a single web page. The reason for this is that (again, for security reasons) there is no simple way to pass information between web pages. This means that code which is running in the context of one web page cannot communicate with code which is running in the context of another web page. This makes it difficult to create a web based HMI which consists of multiple pages.

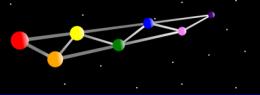
A single web page does not however restrict you to a single HMI "screen". It is possible to create multiple HMI "screens" on a single web page and then hide all but one screen at a time. Each screen can then be revealed one at a time under program control, giving a form of navigation which is more like a traditional non-web application.



6.5.5.4.5 CSS Styling

6.5.5.4.5.1 Basic Page Layout

The CSS style sheet for this example is shown below. The CSS styles used in your own project will be different, depending on what you want the display to look like. The important thing to notice here is how the "display" properties for mainscreen, eventscreen, and alarmscreen are set. "display" for mainscreen is set to "block" while the others (eventscreen and alarmscreen) are set to "none". This means the main screen will be displayed when the page is loaded, while the others will be hidden. These values are then changed through Javascript to hide or reveal screens as required. Also, a fixed-width paged design is used to control page presentation to make layout of graphical screens easier.



```

/* Page layout CSS for the HMI minimal demo demo.
   This contains the CSS which controls the overall page layout
   and appearance, as well as the display of the navigation menu
   and the display of the different "screens".
 */

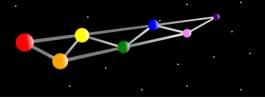
/* This controls the appearance of the area outside of the defined
screens. */
body,
html {
    margin:0px;
    padding:0px;
    background:white;
    color:black;
}
/* This provides the background image. */
body {
    min-width:1000px;
    min-height:600px;
}

/* This provides a filler strip between the nav menu and the main
display area. */
#filler {
    background:tomato;
    margin:0px auto;
    width:1000px;
}

/* This handles the navigation (menu) bar. */
#nav {
    background:tomato;
    font-size: 150%;
    margin-top:0px;
    margin-bottom:0px;
    padding-bottom:2px;
}
#nav ul{
    padding-top:5px;
    padding-bottom:5px;
    list-style:none;
}
#nav li{
    display:inline;
}

#nav a {
    float:left;
    text-decoration:underline;
    color:white;
    background-color:grey;
    border-right:2px solid white;
    padding-top:2px;
    padding-bottom:2px;
}

```



```

        padding-left:5px;
        padding-right:5px;
    }
    #nav a:hover {
        background-color:green
    }

    /* This provides a filler strip at the bottom of the main display
area. */
    #footer {
        background:tomato;
        clear:both;
    }
    #footer p {
        padding:5px;
        margin:0px;
    }

    /* This controls display of the main screen. */
    #mainscreen {
        background:bisque;
        width:1000px;
        display: block;
    }

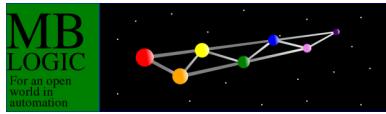
    #mainscreen h2, #mainscreen h3, #mainscreen p {
        padding:0px 10px;
    }

    /* This controls display of the events screen. */
    #eventscreen {
        background:bisque;
        width:1000px;
        display: none;
    }
    #eventscreen h2, #eventscreen h3, #eventscreen p {
        padding:0px 10px;
    }

    /* This controls display of the alarms screen. */
    #alarmscreen {
        background:bisque;
        width:1000px;
        display: none;
    }
    #alarmscreen h2, #alarmscreen h3, #alarmscreen p {
        padding:0px 10px;
    }

```

6.5.5.4.5.2 SVG Button Animation with CSS



The SVG push buttons can be "animated" when they are "clicked" to give immediate feedback to the user. The easiest way to do this is through CSS styling. In the example below, we reverse the "stroke" property from black to white when the button is activated (clicked). This gives a very effective "flash" to the outline and lettering on the button without requiring any scripting. Other properties can also be used to give different effects.

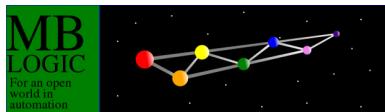
An alternate method using Javascript is described in the section on the display libraries.

This example also uses "hover" to alter the stroke width when the mouse cursor hovers over the button. This gives the operator the "feel" that this object is special and is active.

```
/* The following style is used to animate SVG push buttons. It causes
the
        outline (stroke) of the button to reverse when activated. */
.buttonactivate {
    stroke: black;
    stroke-width: 5px;
}

.buttonactivate:hover {
    stroke: black;
    stroke-width: 7px;
}

.buttonactivate:active {
    stroke: white;
    stroke-width: 5px;
}
```



6.5.5.5 HMI Javascript

6.5.5.5.1 Overview

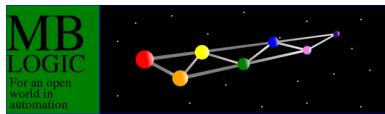
Javascript is a programming language which is supported by almost all modern web browsers. Despite the name, Javascript has no relationship to the Java programming language. They are two different languages, which both happen to have a syntax that was largely inspired by 'C'.

Javascript belongs to what are sometimes referred to as "scripting" languages. A scripting language is a language which is intended for writing applications, as opposed to "systems" languages, which are intended for writing things like operating systems or compilers. Like most scripting languages, Javascript has many powerful features based on object oriented programming concepts. The details of Javascript programming are not discussed here. There are many books and web sites devoted to Javascript programming which should be consulted for that information.

6.5.5.5.2 Javascript and the HMI System

The HMI system makes use of Javascript to create what is often called an "AJAX" application. "AJAX" stands for "Asynchronous Javascript And XML". The history and philosophy of AJAX applications won't be discussed here, other than to mention that there are many books and web sites devoted to the subject. The web based HMI is not a closed system, in that AJAX components from popular AJAX libraries may be used to provide additional features if desired.

Like other AJAX applications, the HMI system relies on Javascript to provide two things. It provides a communications channel to the server which operates outside of the normal page load mechanism, and it is used to script page features to allow the HMI to be manipulated without having to load a new page from the server. Both of these features provide a more interactive application which is more like a conventional local application, and less like a traditional web page.



6.5.5.3 HMI Communications Library

The communications protocol used by the HMI system is implemented in a client version called "libhmiclient.js". This protocol library allows the client (web browser) to communicate with the server without having to reload the web page. This permits the HMI system to poll the server on a regular basis (e.g. once per second) to receive updates about inputs, events, alarms, and other HMI information. This makes the libhmiclient.js library an import part of the HMI client.

The HMI communications library makes use of a third party library called "json2.js" to provide the JSON encoding and decoding. "json2.js" is a standard library from the official JSON website at "www.json.org", and is licensed as public domain.

Details of how to use the HMI communications library may be found in another section.

6.5.5.4 HMI Javascript Library

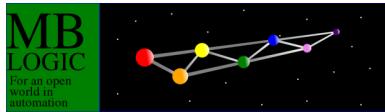
The HMI has a standard Javascript library called "libmbhmi.js" intended for helping to build HMI applications. It includes functions for things such as manipulating the SVG, controlling which "screen" is visible, and event and alarm handling.

6.5.5.5 In-Page Javascript

In addition to the external standard libraries, a typical application may use Javascript embedded directly in the web page. This could be used for code which is uniquely part of the application, as opposed to reusable libraries. This may include things such as initialision code, custom data manipulation, and the main polling loop.

6.5.5.6 Other Uses for Javascript

In addition to the above, external Javascript files may be used to hold application data, as opposed to normal Javascript code. Some examples of where this may be used are



Javascript object literals to hold event and alarm messages, and the messages for text displays. An example of this is shown below.

```
/*
Event message texts for HMI demo.
24-Oct-2008

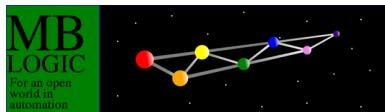
*

event_text = {
"PumpRunning" : "Tank pump is running.",
"PumpStopped" : "Tank pump is stopped.",
"Tank1Empty" : "Tank 1 is empty.",
"Tank1Full" : "Tank 1 is full.",
"Tank2Empty" : "Tank 2 is empty.",
"Tank2Full" : "Tank 2 is full."
}

alarm_text = {
"PB1Alarm" : "PB1 was pressed.",
"PB2Alarm" : "PB2 was pressed.",
"PB3Alarm" : "PB3 was pressed.",
"PB4Alarm" : "PB4 was pressed."
}

// This is for text indicators.
Msg_HMIDemoColours = ["red", "orange", "yellow", "green", "blue",
"indigo", "violet"]
```

These text messages can be accessed directly by the Javascript code in the application, and can be passed to the standard libraries for such things as event and alarm handling. Placing the text messages in a separate file is usually more convenient and easier to maintain than placing them directly in the web page itself.



6.5.5.6 HMI SVG Graphics Elements

6.5.5.6.1 Overview

SVG stands for *Scalable Vector Graphics*. SVG is one of the [web standards](#) specified by the W3C, which is the group responsible for defining and documenting the standards used by modern web browsers. SVG allows high quality interactive graphics to be embedded directly into a web page.

Unlike simple PNG or JPEG graphics, SVG is an XML based description of a graphic element which can be directly manipulated by client scripting. That is, the SVG graphic or image can be changed dynamically without having to reload the web page or download a new image. This allows graphical web based HMI applications to be created which have the sort of interactive graphics normally associated with conventional HMI programs.

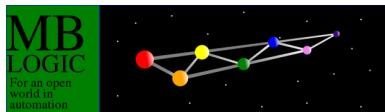
6.5.5.6.2 SVG and Industrial HMI

SVG can be used for industrial HMI systems in two ways. One is to provide static vector graphics for background images, plans, maps, etc. Because the drawing is a "vector" representation rather than a simple "raster" (also known as a "bit map") image, the SVG graphic can be zoomed and resized without losing any resolution. This makes it suitable for displaying high quality drawings and plans on a web page.

The other use, and the one we will concentrate on here, is using SVG to create HMI symbols which we can manipulate dynamically. These symbols can include such things as pilot lights, gauges, tanks, pumps, etc. The dynamic manipulations can include such things as changing colours, rotation angles, or simple animations.

6.5.5.6.3 Example 1, A Simple Pilot Light

The following shows an example of a symbol built using a single basic SVG command.



A simple round pilot light can be created as a circle. The SVG XML to create this is shown below.

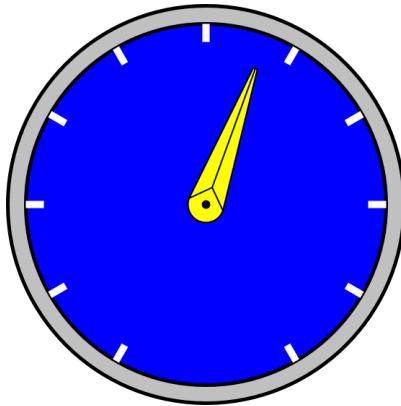
```
<circle id="PL1" cx="25px" cy="25px" r="25px" fill="red"  
stroke="black" stroke-width="5px"/>
```

This creates a circle which we can reference by the name "PL1" (id="PL1"). We can change any characteristic of this circle via Javascript, but for pilot lights we would typically want to change colour. The following example shows how easy this can be.

```
svgdoc.getElementById("PL1").setAttribute('fill', "green");
```

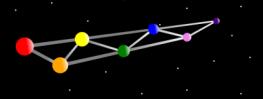
6.5.5.6.4 Example 2, Bulding a Complex Graphic

The following shows how several SVG commands can be combined to create a more complex symbol.



Example 1, a dial gauge. The SVG code to create this is shown below.

```
<!-- This adds the outer ring. -->  
<circle cx="0px" cy="0px" r="275px" fill="silver" stroke="black"  
stroke-width="5px"/>  
<!-- This adds the outer part of the dial background. -->  
<circle cx="0px" cy="0px" r="250px" fill="blue" stroke="black" stroke-  
width="5px"/>  
<!-- These add the "tick" marks to the dial. -->
```



```

<g stroke="white" stroke-width="10px">
    <line x1="0" y1="-250" x2="0" y2="0"/>
    <g transform="rotate(30)">
        <line x1="0" y1="-250" x2="0" y2="250"/>
    </g>
    <g transform="rotate(60)">
        <line x1="0" y1="-250" x2="0" y2="250"/>
    </g>
    <g transform="rotate(90)">
        <line x1="0" y1="-250" x2="0" y2="250"/>
    </g>
    <g transform="rotate(120)">
        <line x1="0" y1="-250" x2="0" y2="250"/>
    </g>
    <g transform="rotate(150)">
        <line x1="0" y1="-250" x2="0" y2="250"/>
    </g>
    </g>
    <!-- This covers the inner part of the "tick" marks and
        provides most of the dial background. -->
    <circle cx="0px" cy="0px" r="225px" fill="blue"
        stroke="none" stroke-width="0px"/>

    <!-- This is the centre part of the pointer. -->
    <circle cx="0px" cy="0px" r="25px" fill="yellow" stroke="black"
        stroke-width="2px"/>
    <circle cx="0px" cy="0px" r="5px" fill="black" stroke="black" stroke-
        width="2px"/>

    <!-- This is the actual pointer which we rotate. -->
    <g id="Dial1" transform="rotate(0)">
        <polygon fill="yellow" stroke="black" stroke-width="2px"
            points="-25,0 -3,200 3,200 25,0 0,25" />
        <line x1="0" y1="25" x2="0" y2="200" stroke="black" stroke-
            width="2px"/>
    </g>

```

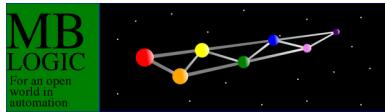
Most of the dial is static SVG which provides "decoration". The part we want to manipulate is the pointer, which we want to rotate. The following Javascript code shows how this can be done.

```

svgdoc.getElementById("Dial1").setAttribute("transform", "rotate(" +
newangle +")");

```

6.5.5.6.5 Visual Effects



SVG allows for visual effects such as colour gradients and drop shadows as shown in the push button below.



The SVG code to create this is shown below.

```
<!-- This shows a push button. -->
<g transform="translate(0,0)"
    onmousedown="MBHMIProtocol.WriteToggleImmediate('PB2',
'PL2');" >

    <!-- The following filter is used to add a drop shadow. -->
    <filter id="PB2dropshadow" >
        <feGaussianBlur stdDeviation="2" >
            </feGaussianBlur >
    </filter >

    <!-- This applies a linear colour gradient across the button.
-->
    <linearGradient id="PB2Gradient" x1="0" y1="1" x2="0" y2="0" >
        <stop offset="50%" stop-color="red" />
        <stop offset="100%" stop-color="white" />
    </linearGradient >

    <!-- This rectangle is used for the drop shadow and needs to
match
        the size used for the button. -->
    <rect x="7" y="7" width="75" height="75" rx="15"
        fill="grey" stroke="none" stroke-width="0px"
        filter="url(#PB2dropshadow)"/ >

    <!-- This is the actual button. The button "class" connects
it with the
        css style which animates it. -->
    <g class="svgbuttondef" >
        <rect x="0" y="0" width="75" height="75" rx="15"
            fill="url(#PB2Gradient)" stroke-width="5px"/ >

        <!-- This is the text label. -->
        <text x="15" y="45" font-size="24" > PB2</text >
    </g >
</g >
```

6.5.5.6.6 CSS Animation

SVG can be styled using CSS just like ordinary HTML. This means that simple animation effects can be achieved without requiring any scripting. If we apply the following style to the push button in the example above, then when we click on the button the outline of the square and the lettering inside the button (which are both affected by the "stroke" property) turn white. When we release the button, the "strokes" turn back to black. This is a simple way of providing immediate feed back to the operator when they click on a button.

```
/* The following style is used to animate SVG push buttons. It
causes the
outline (stroke) of the button to reverse when
activated. */
.svgbuttondef {
    stroke: black;
}

.svgbuttondef:active {
    stroke: white;
}
```

Button when normal:



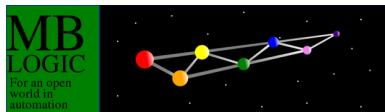
Button when clicked:



CSS animation in SVG appears to work reliably with most web browsers. If there is some reason for not using CSS animation, then some simple Javascript animation can produce an equivalent result.

6.5.5.6.7 More Examples

The following are more examples of the types of graphics that can be created.



6.5.5.6.8 Basic SVG Elements

The complete SVG standard can be found at the link mentioned above. The following however provides a simple overview of some of the basic shapes.

6.5.5.6.8.1 Line

A line is the most basic shape. The following shows a simple line. The line extends from coordinates (x₁,y₁) to (x₂,y₂).

```
<line x1="0" y1="50" x2="150" y2="50"/>
```

6.5.5.6.8.2 Polyline

A polyline is like a line, except it is specified as a series of (x,y) points. There is no limit to the number of points which can be specified.

```
<polyline points="0,0 0,50 140,50" />
```

6.5.5.6.8.3 Polygon

A polygon is like a polyline, but forms a closed shape. The following is used to form a hexagon.

```
<polygon points="0,0 -12,25 0,50 25,50 37,25 25,0" />
```

6.5.5.6.8.4 Rectangle

A rectangle is specified with the upper left corner at position (x,y), a width, and a height. The radius of the corners can also be specified.



```
<rect x="7" y="7" width="150" height="50" rx="15"/>
```

6.5.5.6.8.5 Circle

A circle is specified with the x and y coordinates of its centre, and a radius.

```
<circle cx="25px" cy="25px" r="25px"/>
```

6.5.5.6.8.6 Text

Text can be displayed by specifying the (x,y) coordinates, and the text which is desired.

```
<text x="50" y="30" >Some sample text</text>
```

6.5.5.6.8.7 Stroke and Fill

Stroke refers to the outline of an element, or in the case of text, to the width of the lines making up the characters. The colour of the stroke can be set using the "stroke" property. The width of the stroke can be set using "stroke-width". Stroke width is normally specified in "pixels" (px).

Fill refers to the colour which fills an enclosed shape. This can be set by using "fill".

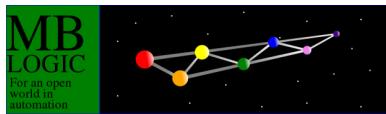
Font size in text can be modified by using "font-size".

```
<circle cx="25px" cy="25px" r="25px" fill="red"
        stroke="black" stroke-width="5px"/>

<text x="50" y="30" font-size="24">Some sample text</text>
```

As well as specifying them directly, properties can also be set by referring to a definition located elsewhere. For example, the following shows the fill being defined by referring to a linear gradient definition (not shown here). The reference is in the form "url(#SomeDefinition)".

```
<rect x="7" y="7" width="150" height="50" rx="15"
fill="url(#PB2Gradient)"/>
```



6.5.5.6.8.8 *Comments*

SVG comments follow the same format as HTML comments.

```
<!-- This is a comment. -->
```

6.5.5.6.8.9 *SVG Document Definitions*

Like HTML, SVG requires that the web browser be informed what sort of mark up the content is. That is, the web browser has to be told to expect SVG. This allows blocks of SVG to be mixed in with HTML in the same web page.

The following defines a block of SVG, with a canvas area 250 pixels wide and 60 pixels high. It then draws a rectangle in that area and closes off the block with an SVG close tag.

```
<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:html="http://www.w3.org/1999/xhtml" width="250px"
height="60">

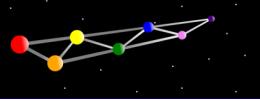
    <!-- Display a rectangle. -->
    <rect x="7" y="7" width="150" height="35" rx="15"
          fill="red" stroke="black" stroke-width="5px" />

</svg>
```

6.5.5.6.9 *SVG Libraries*

SVG elements can be collected into groups and re-used rather than having to repeat the same text over and over again. The following shows a simple example.

First we will define a few colours. The details of how these work will be explained elsewhere, but it is sufficient to say that these provide a "drop shadow" (a blurred background shadow effect) and a silver colour "gradient" (a colour which gradually shades into another - silver into white in this case). These colours can be addressed by using their "id". These are "MB_DropShadowFilter" in the case of the drop shadow,



and "MB_SilverGradient" in the case of the silver colour gradient. Note that these elaborate colour effects are not strictly necessary, but this does show how such decorative effects can be re-used.

```
<!-- These are some colour definitions which are used below. -->
<defs>
    <!-- The following filter is used to add a drop shadow. -->
    <filter id="MB_DropShadowFilter">
        <feGaussianBlur stdDeviation="2">
        </feGaussianBlur>
    </filter>

    <!-- This provides a silver shading colour. -->
    <linearGradient id="MB_SilverGradient"
        x1="1" y1="1" x2="0" y2="0">
        <stop offset="50%" stop-color="silver" />
        <stop offset="100%" stop-color="white" />
    </linearGradient>

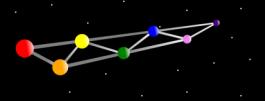
</defs>
```

Next, we define some geometric shapes. The first is a a decorative nut which we call "MB_NUT". Again, this decorative nut can be addressed by using its "id" (id="MB_NUT"). This nut is made from two octagonal polygons. The first polygon provides a background shadow which is slightly offset from the actual nut itself, which is formed from the second polygon.

Here we use the two colours which we defined above. The shadow is given a blurred background effect using the colour filter="url(#MB_DropShadowFilter)". The nut itself is given a shaded silver effect using the colour fill="url(#MB_SilverGradient)".

Below that we define the actual pilot light which we identify as "MB_PilotLightRound". The actual pilot light is just a simple circle which sits on top of the nut we defined above. The same nut can also be used in additional widgets.

```
<!-- Define the actual Pilot lights. -->
<defs>
    <!-- This is a decorative octagonal nut. -->
    <g id="MB_NUT">
        <polygon transform="translate(5,5)" fill="grey"
stroke="none"
            filter="url(#MB_DropShadowFilter)">
```



```

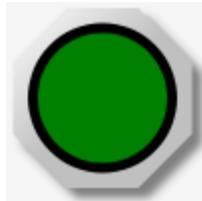
    points="18,-45 -18,-45 -45,-18 -45,18 -18,45
18,45 45,18 45,-18 18,-45" />
    <polygon fill="url(#MB_SilverGradient)" stroke="none"
    points="18,-45 -18,-45 -45,-18 -45,18 -18,45
18,45 45,18 45,-18 18,-45" />
</g>

    <!-- Circular pilot light. r = 35 px -->
<g id="MB_PilotLightRound">
    <!-- This is a decorative nut. -->
    <use xlink:href="#MB_NUT"/>
    <!-- This is the part which changes colour. -->
    <circle cx="0px" cy="0px" r="35px" stroke="black"
stroke-width="5px"/>
</g>

    <!-- We can defined more things here if we wish. -->
</defs>

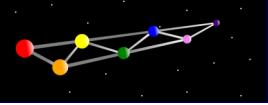
```

This gives us a pilot light which looks something like the following (although the actual colour of the centre of the pilot light is undefined at this point).



Next, we use the pilot light we have defined. In this example we use it three times to create pilot lights "PL1", "PL2", and "PL3". Each of these pilot lights has three components.

- First we position it using transform statements (e.g. `transform="translate(835,110)"`). This simply positions it on the page using X and Y coordinates.
- Next we give each one a unique name (e.g. `id="PL1"`). These names are used elsewhere (not shown here) to connect the pilot lights to their data sources and to animate them.
- Finally, we call the SVG widgets we defined above (e.g. `use xlink:href="#MB_PilotLightRound"`). The same definition can be reused as many times as desired. It is the "id" which distinguishes one instance from another.



```

<!-- ... Much further down the page ... -->

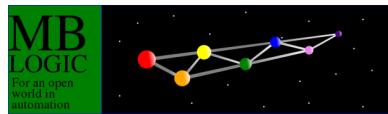
<!-- Now we use the pilot light we defined above. -->
<g transform="translate(835,110)">
    <g id="PL1">
        <use xlink:href="#MB_PilotLightRound" />
    </g>
</g>

<!-- And we use it again for another pilot light. -->
<g transform="translate(835,210)">
    <g id="PL2">
        <use xlink:href="#MB_PilotLightRound" />
    </g>
</g>

<!-- And again for a third one. -->
<g transform="translate(835,310)">
    <g id="PL3">
        <use xlink:href="#MB_PilotLightRound" />
    </g>
</g>

```

A number of common devices such as push buttons, pilot lights, selector switches, etc. are provided with the HMI system. Simply copy the definitions into your own HMI web page and use them as desired. You can also create your own widgets using the same principles.



7 Installation

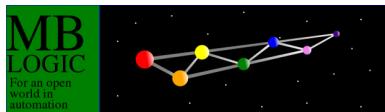
7.1 Overview

Installation consists of the following steps:

1. Extract the application files from the archive.
 2. Obtain and install the required third party software packages.
 3. Test the installation.
-

7.1.1 Help Topics

- [Installing the Base Application](#)
- [Installing Third Party Support Software](#)
- [Starting the Application](#)
- [Testing the Application](#)
- [Stopping \(Shutting Down\) the Application](#)
- [Using Port 502](#)



7.2 Base Application

7.2.1 Overview

The application is distributed as a source archive file. For most platforms (including Linux), this is a "tar.gz" file. For MS-Windows, this is a "zip" file. The MS-Windows version has the special new-line character combination (CR-LF) required by that platform, but is otherwise identical to the standard version.

7.2.2 Installation

1. Select or create a directory into which to extract the application files.
 2. Extract the archive file into the desired location.
 3. Several sub-directories will be created as part of the extraction process.
 4. The application itself will now be installed. However, you cannot run it until you install the third party support software (see the help instructions on how to perform that step).
-

7.2.3 Application Directory Contents

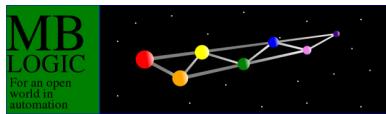
You should see the following items present in the application directory.

7.2.3.1 Application Directory Contents

There will be a directory called "mblogic". It will contain a number of files with ".py" extensions. These are part of the application program itself. These files will be automatically compiled to ".pyc" files when the application first starts. There will also be several sub-directories with additional code files, as well as html web pages for the help system and the status monitorin system. There is nothing in these directories that you will need to change.

7.2.3.2 HMI Directory Contents

There will be a directory called "hmipages". It will contain a number of files with ".xhtml", ".js", ".css", and ".png" extensions. These files constitute a sample HMI



client (web page) that comes with the system. You will replace some of these files with your own versions later if you create your own HMI application.

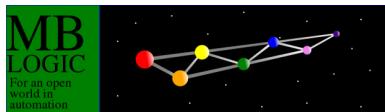
7.2.3.3 Sample Soft Logic Program

There will be several files which are provided as a sample soft logic application. This sample application works with the sample HMI client (web page). These files are:

- "plcprog.txt". This is a sample soft logic program.
- "mbserver.config". This is provided as a sample server communication configuration file. You will need this file to use the sample HMI demo. You will also need it to access the on-line help functions and system status functions until you replace it with your own configuration.
- "mbclient.config". This is provided as a sample client communication configuration file. You will need this file to use the sample HMI demo.
- "mblogic.config". This is provided as a sample soft logic IO configuration file. You will need this file to run the sample soft logic program and to use the HMI demo.
- "mbhmi.config". This is provided as a sample HMI server configuration file. You will need this file to use the HMI demo.

7.2.3.4 Shell Script to Start the System

There will also be a shell script which is used to start the system. For the Linux version, this is called "mblogic.sh". For the MS-Windows version, this is called "mblogic.bat". This file simply calls the main program file in the "mblogic" directory.



7.3 Third Party Support Software

7.3.1 Overview

The system is built on a number of third party packages which must be installed separately before the system will function. These third party packages are:

- Python - Run time language support for the Python programming language.
- Twisted - Networking library.

Several additional packages are required for MS Windows (these are not required for Linux).

- pywin32 - Additional Python library (required for MS Windows only).
- setuptools - Additional library required to install "egg" format libraries (required for additional libraries).
- zope.interface - An additional library required for Twisted (for Twisted versions 10.0.0 or later).

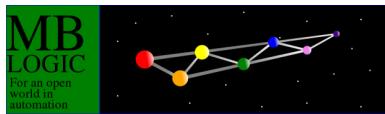
All required third party support packages are free and can be obtained from normal secure internet sites (see below).

The system has been tested with Python versions 2.5 and 2.6. Other versions may work but have not been tested. However, version 2.6 is the recommended version. The version of Twisted installed must be compatible with the version of Python used. If you are using Linux, the version compatibility will be taken care of automatically. If you are using MS Windows, the Twisted web site will indicate which version of Twisted you need for which version of Python.

7.3.2 Obtaining and Installing Third Party Software

The required third party software can be obtained from the following locations.

7.3.2.1 For Linux



For Linux, all required packages can normally be obtained from your distro's package repository. Use your package manager (e.g. Synaptic) to download the packages. Some packages (e.g. Python) may already be installed as part of the standard OS distribution.

Install the following in the order listed:

1. Python - Version 2.6.
2. Twisted. The Debian package name is "python-twisted". You will need python-twisted, python-twisted-core, python-twisted-web (not web2), and any dependencies (your package manager should be able to figure the dependencies out by itself).

7.3.2.2 For MS Windows

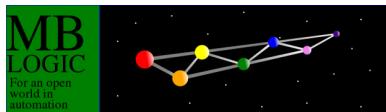
Installing software on MS Windows is traditionally more difficult than with Linux, as MS Windows has no package management or repository system. This means that each package must be downloaded and installed separately. However, this is relatively straight forward if you follow the steps below.

The packages can be obtained from the following locations (don't install them just yet though):

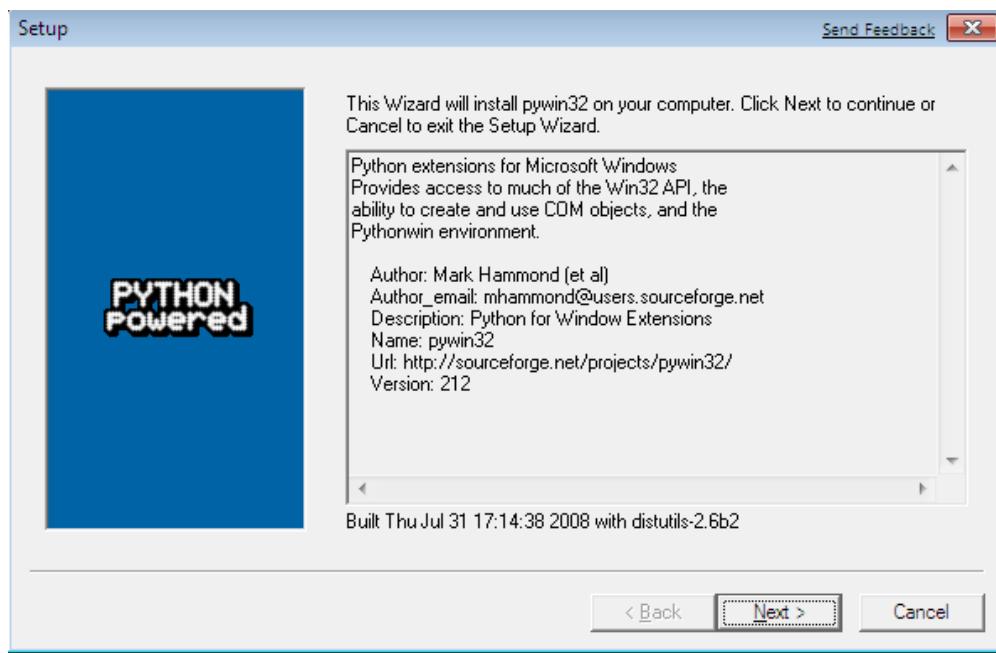
- Python - <http://Python.org>
- pywin32 - <http://sourceforge.net/projects/pywin32/>
- Twisted - <http://twistedmatrix.com/trac/>
- SetupTools - <http://pypi.python.org/pypi/setuptools>
- Zope Interface - <http://pypi.python.org/pypi/zope.interface>

Once you have obtained all the software, install it in the following order:

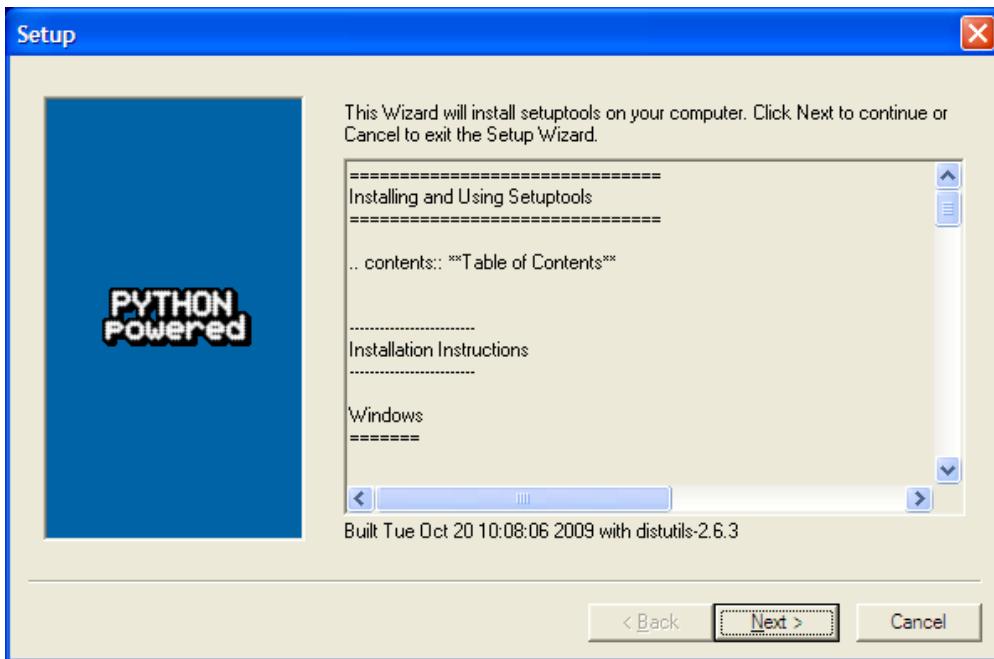
1. **Python.** Make a note of where you installed this, as the other packages will need to find it. (Note, the actual install screen will look slightly different from the one shown below. This screen shot was taken after the install was run once already). Run the installer and follow the instructions in the install menus.



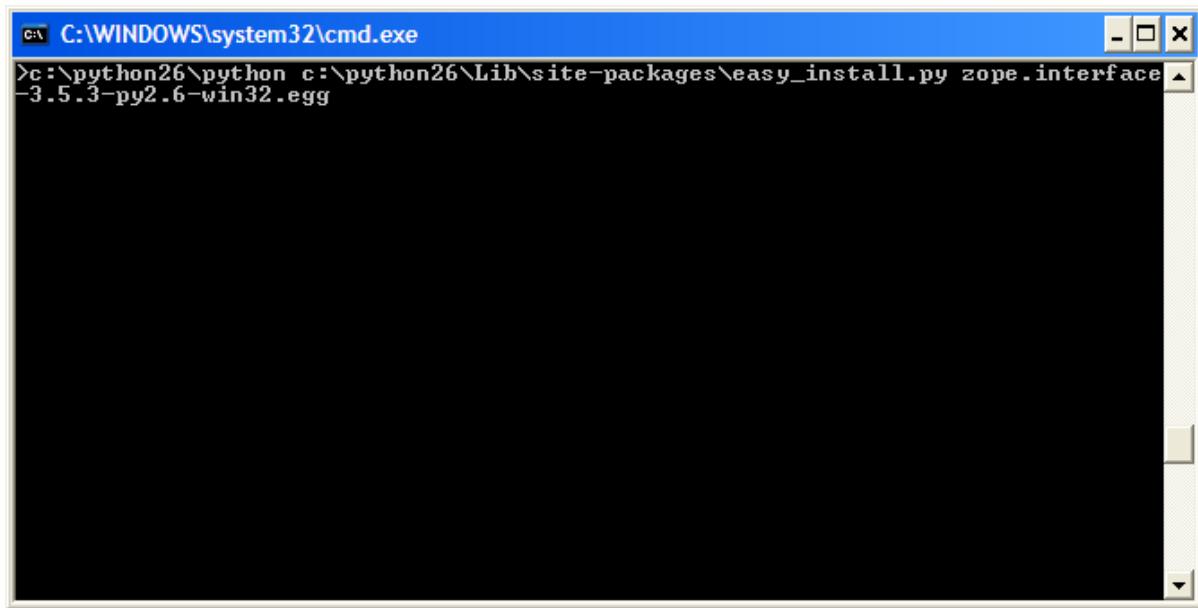
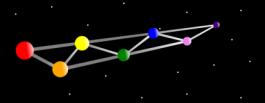
2. **pywin32**. Run the installer and follow the instructions in the install menus.



3. **SetupTools**. Run the installer and follow the instructions in the install menus. This is required only for installing Zope Interface. If Zope Interface is already installed (or you do not require it because you are installing an older version of Twisted), you can skip this step.

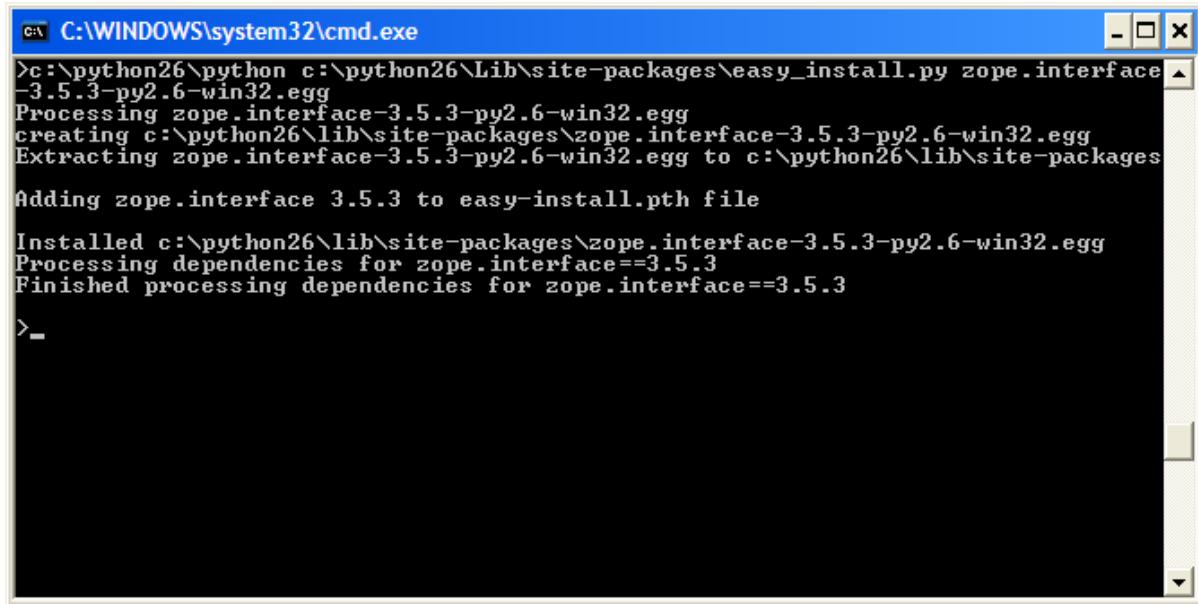


4. **Zope Interface.** This is required for versions of Twisted from 10.0.0 or later. Earlier versions of Twisted package Zope Interface in the Twisted package and do not require Zope Interface to be installed separately. Zope Interface must be installed using the command line. Move the Zope Interface "egg" package to a working directory and open a command line window in the resulting directory. Type "c:\python26\python c:\python26\Lib\site-packages\easy_install.py zope.interface-3.5.3-py2.6-win32.egg" (without the quotes). If you are using a different version of the Zope Interface, the file name "zope.interface-3.5.3-py2.6-win32.egg" may be slightly different. If you have installed Python in a different location, or if you have installed a different version, you will have to alter the "c:\python26" part of the command accordingly. (e.g. "c:\python25\python etc." if you are using Python 2.5).



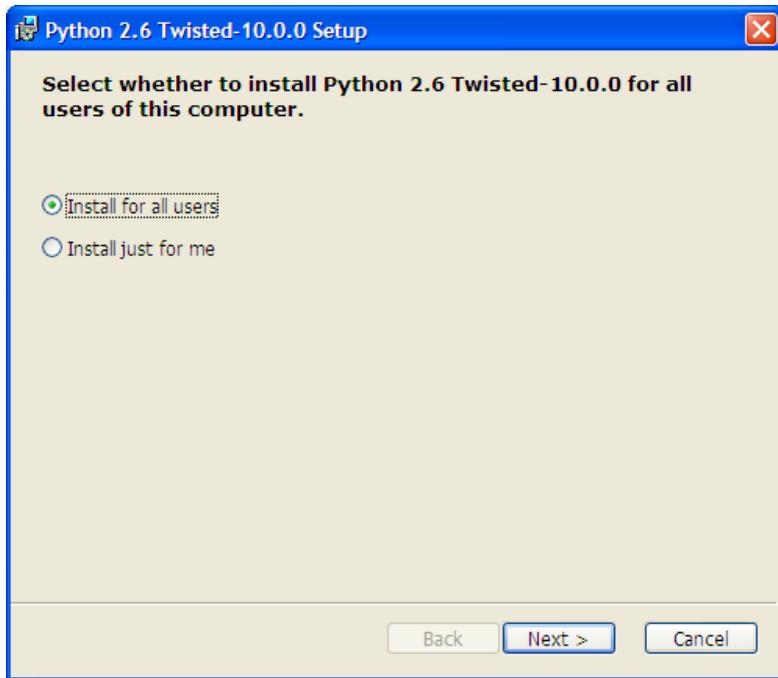
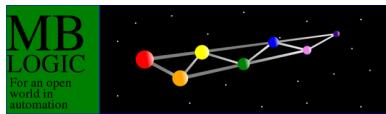
```
C:\WINDOWS\system32\cmd.exe
>c:\python26\python c:\python26\Lib\site-packages\easy_install.py zope.interface-3.5.3-py2.6-win32.egg
```

You should see something like the following as a result. You may see some "deprecation warnings", depending on what version you install. These are not serious and are simply warning of new features or features that will be changed in an upcoming version.



```
C:\WINDOWS\system32\cmd.exe
>c:\python26\python c:\python26\Lib\site-packages\easy_install.py zope.interface-3.5.3-py2.6-win32.egg
Processing zope.interface-3.5.3-py2.6-win32.egg
creating c:\python26\lib\site-packages\zope.interface-3.5.3-py2.6-win32.egg
Extracting zope.interface-3.5.3-py2.6-win32.egg to c:\python26\lib\site-packages
Adding zope.interface 3.5.3 to easy-install.pth file
Installed c:\python26\lib\site-packages\zope.interface-3.5.3-py2.6-win32.egg
Processing dependencies for zope.interface==3.5.3
Finished processing dependencies for zope.interface==3.5.3
>_
```

5. **Twisted**. Run the installer and follow the instructions in the install menus.



7.3.2.3 MS Windows DLL Files

Some copies of MS-Windows may have one or more "dll" files missing which prevent the program from operating correctly. This can happen when dll files are unintentionally removed when uninstalling software or making other changes. One dll in particular to check is "mfc71.dll". This is a standard MS Windows library which is used by many programs. This should be in the "c:\windows\system32" directory. If it is missing, you will need to reinstall it. A search on Google will turn up many solutions for this problem (this is a common problem). This file is not necessary for Linux.

7.3.3 Installing on Mac OS/X

There is no documentation for installing on Mac OS/X. However, newer versions of OS/X may come with Python and Twisted already installed.

7.3.4 Testing Third Party Software



To test if the third party software installation is working, you can do the following (you can skip this step if you wish).

7.3.4.1 For Linux

For Linux, open a command line terminal and type "python". You should see the Python interpreter start up with something like the following:

```
Python 2.6.2 (release26-maint, Apr 19 2009, 01:58:18)
[GCC 4.3.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To test if twisted is installed, type the following into the Python interpreter:

```
import twisted
```

If it is already installed, the Python command problem will appear after a brief delay. If it is not already installed, you should get an error which looks something like the following:

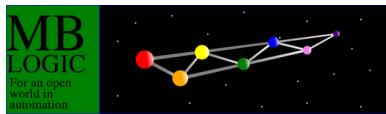
```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named twisted
>>>
```

You should get a result similar to that described for twisted.

To exit Python, press "control-D" (press the control and "d" keys simultaneously). To close the terminal, press "control-D" again.

7.3.4.2 For MS Windows

For MS-Windows, following the instructions given above for Linux, with the following exceptions.



When starting Python, you will need to include the full path name to the Python interpreter. For example: "C:\Python26\python"

Test if twisted is installed correctly by using the following commands:

```
import twisted.internet
```

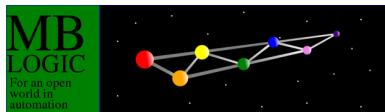
This checks the pywin32 installation as well as twisted.

To exit Python, press "control-Z" (press the control and "z" keys simultaneously).

7.3.5 SimpleJSON

Older versions offered the option of using an improved JSON library called "simplejson". This is no longer used as Python 2.6 has JSON support built directly into its standard library.

If you are using an older version of Python, the system will automatically import a bundled version of "simplejson" but without the performance optimisations of the 'C' version. However, this will make little measureable difference in normal applications.



7.4 Starting the Application

7.4.1 Overview

7.4.2 Starting the Application

When all software has been installed, you can start the application.

7.4.2.1 Starting from Linux

Do **one** of the following"

- If you are using Gnome, double click on the "mblogic.sh" file.
- **Or** from the command line, type "./mblogic.sh".

7.4.2.2 Starting from MS Windows

Do **one** of the following"

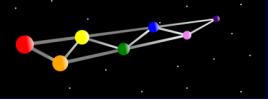
- If you are using the MS-Windows GUI, double click on the "mblogic.bat" file.
- **Or** from the command line, type "mblogic.bat".

The batch file ("mblogic.bat") assumes that you have installed Python version 2.5 or 2.6, and that you have installed it in the default location ("C:\Python25" or "C:\Python26"). If you have a different version of Python installed, or have installed Python in a different location, you will have to edit the batch file accordingly. To edit the batch file, open it with a text editor (e.g. MS Notepad), and change it to match the version and location of your Python installation.

7.4.2.3 Start-up Messages

You should see something like the following appear:

```
Starting MBLogic.  
Starting system status web server...  
Starting ModbusTCP server...  
Starting REST web service...  
Starting help system web server...  
Starting HMI web service...  
Starting ModbusTCP clients...
```



```

Started to connect outgoing client PumpPressure
Started to connect outgoing client FanCommand
Started to connect outgoing client Robot1
Started to connect outgoing client Beacon
Started to connect outgoing client SlidePosition
Started to connect outgoing client Conveyors
Started to connect outgoing client FanSpeed
Started to connect outgoing client Horn
Soft logic system started.

```

Server Loopback Server running at 9:39:30, Mon 05 of Jan, 2009

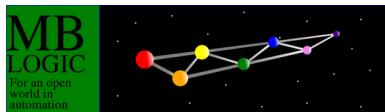
```

...
Incoming client connected to Modbus TCP server from 127.0.0.1.
Incoming client connected to Modbus TCP server from 127.0.0.1.
Incoming client connected to Modbus TCP server from 127.0.0.1.
Incoming client connected to Modbus TCP server from 127.0.0.1.
Incoming client connected to Modbus TCP server from 127.0.0.1.
Incoming client connected to Modbus TCP server from 127.0.0.1.
Incoming client connected to Modbus TCP server from 127.0.0.1.
Incoming client connected to Modbus TCP server from 127.0.0.1.
Connected outgoing client FanCommand.
Connected outgoing client FanSpeed.
Connected outgoing client Horn.
Connected outgoing client Robot1.
Connected outgoing client PumpPressure.
Connected outgoing client Beacon.
Connected outgoing client SlidePosition.
Connected outgoing client Conveyors.

```

Some of the above lines may appear in a different order depending on when the connections are made. The above assumes you are using the example configuration file included with the release. If you are ***not*** using the example configuration, you will see a different series of messages relating to which servers and clients are starting up and connecting.

Port usage is determined by the configuration file.



7.5 Testing the Application

7.5.1 Overview

This section describes how to test the application after you have installed it.

7.5.2 Checking the Web Interface

Check the system status web page by starting your web browser and entering the following URL (this assumes you are using the sample configuration in mbserver.config):

```
http://localhost:8080/statussystem.html
```

You should see a web page titled "System Status" similar to the one shown below.

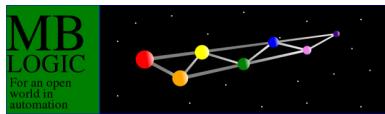
The screenshot shows a web browser window titled "System Status - Shiretoko". The address bar contains the URL "http://localhost:8080/statussystem.html". The main content area has a blue header with the "MBLogic" logo and the tagline "for an open world in automation". Below the header is a navigation menu with links: Home, Configure, Control, Data Table, Monitor, XRef, and Help. The left side of the page is titled "General System Parameters" and lists the following information in a table:

System Name	Loopback Server
Software	MBLogic
Version	07-Jun-2010
Started at	Wed 30 Jun 2010 03:31:14 PM EST
Uptime (hrs)	0.01

The right side of the page is titled "Subsystem Status" and displays a table with the following data:

Servers	6
TCP Clients	8
Generic Clients	1
HMI	Ok
Soft Logic IO	Ok
Soft Logic	Running
Status Client	Ok

Near the right side of the page should be a section titled "Subsystem Status". There should be a table with headings "Servers", "Clients", "Soft Logic", "Soft Logic IO",



and "HMI". Each will be accompanied by a text status message and a coloured background status indication.

The monitoring system uses Javascript to fetch fresh data from the server on a regular basis. The page updates itself on a live basis and Javascript must be enabled to view any data.

7.5.3 System Help

If you can view the status system web page (see above), you will now be able to access the system on-line help. Click on the "Help" link in the menu near the top of the stats web page.

7.5.4 Checking the User Help Server

Check the user help system by starting your web browser and entering the following URL (this assumes you are using the sample configuration in mbserver.config):

```
http://localhost:8081/index.html
```

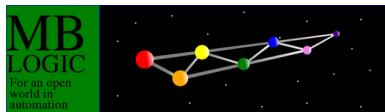
You should see a web page titled "User Help". The default page is a place holder which can be replaced with other content.

7.5.5 Running the HMI Demo

Check the HMI system by starting your web browser and entering the following URL (this assumes you are using the sample configuration in mbserver.config):

```
http://localhost:8082/hmidemo.xhtml
```

Alternatively, the available HMI screens are listed at the bottom of the "configure" page. Click on the web page name to open it.



System Configure - Minefield

File Edit View History Bookmarks Tools Help

http://localhost:8080/statuspages/statusconfigure.html

System Configure

MBLogic for an open world in automation

Home Configure Control Monitor Data System XRef Help

Configure

The following shows the currently running configuration. Click on the sub-system name to view the configuration details.

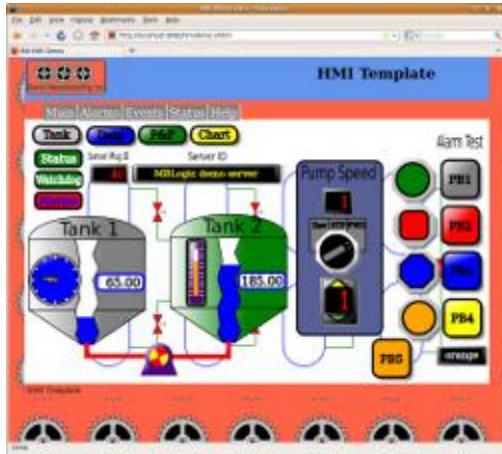
Sub-System	Status	Started At	Signature
Servers	Ok	Tue 19 Oct 2010 12:21:42 AM EST	766dd48d55686d33bfc9cfa55f6fb37
Clients	Ok	Tue 19 Oct 2010 12:21:42 AM EST	c4fad248e403589f80bd20ac8b8b2c8a
HMI	Ok	Tue 19 Oct 2010 12:21:43 AM EST	4f61b27b32a324a7cc71ada3cf048b
Soft Logic IO	Ok	Tue 19 Oct 2010 12:21:43 AM EST	24e6c9f7aba78d5653fa1088838cf6a7
Soft Logic Program	Ok	Tue 19 Oct 2010 12:21:43 AM EST	f9a6df02b6542afa145d0af1abbca18a

HMI Web Pages

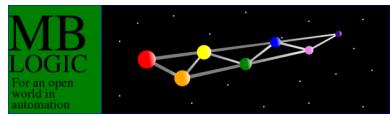
HMI Web Page	Signature
hmidemo.xhtml	0826c99d05785fd5b2560e3bf5acee28
hmidemo2.xhtml	a430f978fcac07c034c37ac351509bbb

System Configure

You should see a web page titled "MB-HMI Demo". You should see a number of graphical elements including push buttons, pilot lights, tanks, and gauges.

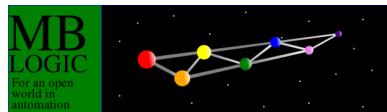


Just above the middle of the screen on the left you should see a numeric display box titled "Server Mesg ID:" the number appearing within the box should be incrementing at a rate of approximately once per second. To the left of this should be a box titled "Comm Status". The background of this box should be green. If you see these items, the system is running correctly.



The HMI demo uses vector graphics based on SVG. Most modern web browsers have this capability, including Firefox, Opera, Apple Safari, Google Chrome, etc. Some browsers based on older technology (e.g. any version of MS Internet Explorer, old versions of Netscape Navigator, etc.) do not have this ability. They may load but not display the page.

The demo page was developed and tested using Firefox, so that browser is recommended for this test. Be sure that you have Javascript enabled, as it is needed to run the communications between the web page and the server.



7.6 Stopping the Application

7.6.1 Overview

This section describes how to shut down the system.

7.6.2 Shut Down Methods

You may shut down the system two different ways. One method is via the systus system interface. The other method is via a keyboard code directly into the terminal window. If your web browser does not fully support the capabilities needed to use the web interface, you will need to use the keyboard command.

7.6.2.1 Using the Menu Interface

To shut down the system via the system status web interface, select the "control" option from the menu. You should see a web page which looks something like the following.

The screenshot shows a web browser window titled "System Control - Minefield". The address bar shows the URL <http://localhost:8080/statuspages/statuscontrol.html>. The main content area is titled "Control". It displays a status summary with four categories: System, Status, Select, Communications, HMI, Soft Logic IO, and Soft Logic. The "Soft Logic" category has a red background and is labeled "1 errors". Below this is a "Control" section with "Reload File" and "Refresh Page" buttons. At the bottom, there are "Restart System" and "Shutdown System" buttons. A "Soft Logic Program Errors" section shows one error: "Error in Events network 6 - missing or incorrect parameters for OUT X25". The footer says "System Control".



Click on the "Shut Down System" button which is located near the bottom of the page. You should see the page change and a confirmation dialogue appear. Click the "Shut Down" button to confirm the shut down. A shut down message will appear, and then automatically clear.



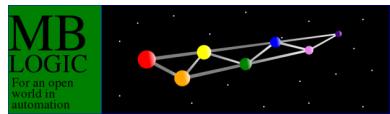
The web page will remain visible because the web browser runs independently of the system itself. However, once the system is shut down you will not be able to update the page or load new web pages because the server is no longer running.

7.6.2.2 Directly Through the Keyboard

To shut down the application using a keyboard command, press "control-C" (press the "control" and "C" keys simultaneously) in the terminal window where the application is running. This will send a signal to the system asking it to shut down.

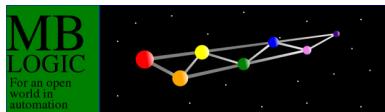
7.6.3 Shut Down Messages

When you have issued a shutdown command, You should see something like the following appear.



```
Operator terminated system at Sat Jan 10 20:45:42 2009
```

Following that you will see a series of lines reporting that the client connections have been lost. This is just the Modbus/TCP server reporting the client connections closing down (which is expected).



7.7 Using Port 502

7.7.1 Overview:

Standard Ethernet protocols use what are called "ports". These are numbers which are part of the Ethernet packets which are used to route them to the correct application program. Both ends of the connection (client and server) have to use an agreed upon port number in order to communicate with each other. The server must "bind to the port number" (request the number from the operating system) before it can listen on that port for requests. Clients on the other hand do not have to bind to a port and are free to send requests to any port.

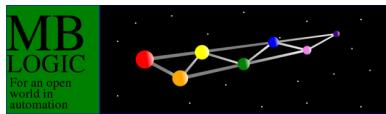
7.7.2 Standard Modbus Port

The standard port number for Modbus/TCP is 502. It is possible to use a different port, provided both client and server can be set to use a different port number. In some cases, this isn't possible, and in most cases other cases it is most convenient to just use the standard port number.

With most operating systems, ports numbers less than 1024 are reserved for standard system services such as e-mail servers, web servers, etc. and are not available to ordinary applications. This poses a problem with Modbus/TCP. Either the Modbus/TCP server needs to gain (temporarily at least) elevated privileges, or else the incoming messages on port 502 need to be re-routed to a different port. The second choice is often the simplest, and poses the fewest security risks.

However, it is important to note that if you ***don't*** need to use the Modbus server, or if you ***do*** need it but can run it on an alternate port number, then you don't need to redirect the port. If the remote client is capable of using an alternate port, that is usually the best solution.

7.7.3 Redirecting Port 502 on Linux



Linux has a built-in facility called "iptables" which can be used to block, re-route and redirect communications. Redirecting traffic arriving on port 502 to a different can easily be done by using iptables. For example, to redirect incoming traffic on port 502 to port 8502:

```
iptables -t nat -I PREROUTING -p tcp --dport 502 -j REDIRECT --to-port  
8502
```

Depending upon how security is set up on your distro, you may need to either log in as "root", or (preferably) use "sudo". For example:

```
sudo iptables -t nat -I PREROUTING -p tcp --dport 502 -j REDIRECT --  
to-port 8502
```

To save this change permanently so that it is automatically loaded when the computer boots up (assuming you use sudo and have nano installed):

```
iptables -t nat -I PREROUTING -p tcp --dport 502 -j REDIRECT --to-port  
8502  
  
sudo sh -c "iptables-save > /etc/iptables.rules"  
  
sudo nano /etc/network/interfaces
```

The above example ends with starting the nano editor (you can use a different editor if you wish) to edit the "interfaces" file. This file stores the Ethernet configuration. Add the following line to the end of the section for the Ethernet port which will be used for Modbus/TCP (typically "eth0").

```
pre-up iptables-restore < /etc/iptables.rules
```

Save the configuration file and exit nano. Now Ethernet packets coming from **outside** the computer to port 502 will be redirected to port 8502. However, packets originating **inside** the same computer (e.g. 'localhost') will not be redirected. Anything originating on the same computer will need to be sent to port 8502.

7.7.4 Port 502 on Microsoft Windows

Microsoft Windows does not offer any built-in security for system ports, so port 502 can be used directly without redirection. However, if you are using any server (on any port), you may need to adjust the firewall (if one is installed) to allow incoming connections.