MAQ®20

Industrial Data Acquisition and Control System

MA1070

MAQ®20 C API User Manual

DATAFORTH®

MAQ®20 C API User Manual
MA1070 Rev. A – March 2024

ISO9001:2015-Registered QMS

Table of Contents

About Dataforth Corporation

"Our passion at Dataforth Corporation is designing, manufacturing, and marketing the best possible signal conditioning, data acquisition, and data communication products. Our mission is to set new standards of product quality, performance, and customer service." Dataforth Corporation, with more than a quarter century of experience, is the worldwide leader in Instrument Class® Industrial Electronics – rugged, high performance signal conditioning, data acquisition, and data communication products that play a vital role in maintaining the integrity of industrial automation, data acquisition, and quality assurance systems. Our products directly connect to most industrial sensors and protect valuable measurement and control signals and equipment from the dangerous and degrading effects of noise, transient power surges, internal ground loops, and other hazards present in industrial environments.

Dataforth spans the globe with more than 50 International Distributors and US Representative Companies. Our customers benefit from a team of over 130 salespeople highly trained in the application of precision products for industrial markets. In addition, we have a team of application engineers in our Tucson factory ready to solve any in-depth application questions. Upon receipt of an RFQ or order, our Customer Service Department provides fast one-day delivery information turnaround. We maintain an ample inventory that allows small quantity orders to be shipped from stock.

Dataforth operates under an ISO9001:2015 quality management system.

Contacting Dataforth Corporation

Contact Method	Contact Information
E-Mail: Technical Support	support@dataforth.com
Website:	www.dataforth.com
Phone:	+1-520-741-1404 and toll free +1-800-444-7644
Fax:	+1-520-741-0762
Mail:	Dataforth Corporation 3331 E. Hemisphere Loop Tucson, AZ 85706 USA

Errata Sheets

Refer to the Technical Support area of Dataforth's website (www.dataforth.com) for any errata information on this product.

1.0 System Features

The MAQ®20 Data Acquisition System encompasses more than 25 years of design excellence in the process control industry. It is a family of high performance, DIN rail mounted, programmable, multi-channel, industrially rugged signal conditioning I/O and communications modules.

Instrument Class Performance

- ±0.035% Accuracy
- Industry leading ±0.3°C CJC Accuracy over full operating temperature range
- Ultra-low Zero and Span Tempco
- Over-range on one channel does not affect other channels
- 1500Vrms Channel-to-Bus Isolation
- 240Vrms Continuous Field I/O Protection
- ANSI/IEEE C37.90.1 Transient Protection
- Ventilated Communications and I/O Modules
- Industrial Operating Temperature of -40°C to +85°C
- Wide Power Supply Range of 7-34VDC
- CE Compliant
- UL/cUL (Class I, Div 2, Groups A, B, C, D) Compliant, file E232858
- ATEX Compliance pending

Industry Leading Functionality

- The system is a Modbus Server and can be operated remotely with no local PC
- Up to 4GB of logged data can be transferred via FTP during real-time acquisition
- Up to 24 I/O modules, or 384 channels, per system, per 19” rack width
- Per-channel configurable for range, alarms, and other functions
- Backbone mounts within DIN rail and distributes power and communications
- System firmware automatically registers the installation and removal of I/O modules
- I/O modules can be mounted remotely from the Communications Module
- Equal load sharing power supply modules allow for system expansion
- Hot Swappable I/O modules with Field-side pluggable terminal blocks on most models
- Sophisticated package enables high density mounting in 3U increments
- DIN Rail can be mounted on a continuous flat panel or plate

Distributed Processing Enables Even More Functionality

- Output modules are programmable for user-defined waveforms
- Discrete I/O modules have seven high level functions:
  - Pulse Counter
  - Frequency Counter
  - Waveform Measurement
  - Time Between Events
  - Frequency Generator
  - PWM Generator
  - One-Shot Pulse Generator

Multiple Software Options

- Free Configuration Software
  - ReDAQ Shape Graphical HMI Design & Runtime Solution
- Intuitive Graphical Control Software
  - ReDAQ Shape Graphical HMI Design & Runtime Solution
  - Python API
  - C API
  - OPC Server
  - LabView VIs

2.0 System Description and Documentation

A MAQ®20 Data Acquisition System must have as a minimum a Communications Module, a Backbone, and one I/O Module. Examples include:

MAQ20-COMx Communications Module with Ethernet, USB and RS-232 or RS-485 Interface
MAQ20-DIOx Discrete Input / Output Module
MAQ20-xTC Type x Thermocouple Input Module
MAQ20-mVxN, -VxN Voltage Input Module
MAQ20-IxN Process Current Input Module
MAQ20-IO, -VO Process Current Output and Process Voltage Output Module
MAQ20-BKPLx x Channel System Backbone

Refer to www.dataforth.com/maq20.aspx for a complete listing of available modules and accessories.

System power is connected to the Communications Module, which in turn powers the I/O modules. For systems with power supply requirements exceeding what the Communications Module can provide, the MAQ20-PWR3 Power Supply module is used to provide additional power. When a MAQ®20 I/O module is inserted into a system, module registration occurs automatically, data acquisition starts, and data is stored locally in the module. The system is based on a Modbus compatible memory map for easy access to acquired data, configuration settings, and alarm limits. Information is stored in consistent locations from module to module for ease of use and system design.

MAQ®20 modules are designed for installation in Class I, Division 2 hazardous locations and have a high level of immunity to environmental noise commonly present in heavy industrial environments.

MAQ®20 communications modules provide connection between a host computer and a MAQ®20 Data Acquisition System over Ethernet, USB, RS-485 or RS-232. Ethernet communications use the Modbus TCP protocol, USB communications are based on the Modbus RTU protocol, and RS-485 and RS-232 communications use the Modbus RTU protocol. Serial communications over RS-485 can be either 2-wire or 4-wire. Each MAQ20-COMx module can interface to up to 24 MAQ®20 I/O modules in any combination allowing high channel counts and great flexibility in system configuration. A removable microSD card can be used by the MAQ20-COMx module to log data acquired from the MAQ®20 I/O modules.

For details on hardware installation, configuration, and system operation, refer to the manuals and software available for download from www.dataforth.com/maq20_download.aspx. This includes, but is not limited to:

MA1036 MAQ®20 Quick Start Guide
MA1040 MAQ®20 Communications Module Hardware User Manual
MA1041 MAQ®20 millivolt, Volt and Current Input Module Hardware User Manual
MA1038 MAQ®20 ReDAQ Shape for MAQ20 User Manual
MA1064 MAQ®20 Python API User Manual
MAQ20-940/-941 ReDAQ Shape Software for MAQ®20 – Developer Version/User Version
MAQ20-952 IPEMotion Software for MAQ®20

3.0 Introduction

The MAQ®20 C API uses an object-oriented approach for communicating with MAQ®20 systems, which provides an intuitive interface where low-level Modbus commands are hidden from normal use. Users can focus on solving the measurement problems at hand, instead of re-inventing how to communicate with modules.

What the C API does for you:

- Communication to MAQ®20 systems from a host PC.

- Address offsetting: When a module is registered in a system, addresses are offset by 2000 * R, where R is the Registration Number. The API allows for absolute addressing and relative addressing for systems and modules. For module relative addressing methods, 0 to 1999 are valid addresses.

- Counts to engineering units: The API can convert engineering units to and from raw counts that the MAQ®20 system uses for representing channel data.

- Mechanisms for freeing memory used by system and module instances.

4.0 Usage and Source Files

Example Visual Studios Project

The Example folder contains a Microsoft Visual Studios 2022 project named Example.sln. This project is provided for demonstration purposes and for quick usage of the MAQ®20 C API without any setup required. The C API uses libmodbus and libcurl for backend implementation. These external library dependencies and all others required by the C API are present within the Example folder and require no additional configurations for the project to build successfully. Additional information regarding these external libraries can be found at the following links:

- libcurl: https://curl.se/libcurl/

- libmodbus: https://github.com/stephane/libmodbus/actions

Adding C API Functionality to New or Existing Visual Studios Project

The API Source Files folder contains all dynamic link libraries(.dll), static libraries (.lib), and header files (.h) required for usage of the C API in different development environments, or in an existing Visual Studios project.

To begin development with the C API the user will need to point the linker and compiler to the location of the .dll, .lib, and .h files. This process will vary depending on the environement and development tools used. The following process was implemented in Visual Studios 2022 using detailed instructions provided Microsoft:

1. Point the linker and compiler to the .h files used by the MAQ®20 C API.

2. Point the linker to the .lib files.

3. Either manually copy over the .dll files into your project, or add a post-build step to automatically copy them.

The general steps outlined above were implemented in the provided Example.sln project. Settings in this project can be used as a reference if needed.

5.0 USB Functionality

TCP is the default communication method used by the MAQ®20 C API, with USB and serial also being supported options. For USB communication to a MAQ®20 system a USBXpress USB to UART driver is required. In some cases, the USB to UART driver can be overridden by a Windows automatic install of a different USBXpress driver. If this occurs, proceed with the following instructions for installation of the correct driver:

1. Download the correct driver from Silicon Labs' website: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers.

2. In Windows Device Manager, uninstall the existing USBXpress driver if the MAQ®20 is being detected within the USB drop-down instead of the COM Port drop-down.

3. The Silicon Labs CP210x device should now appear under the "Other Devices" drop-down section.

4. Right click on the displayed CP210x device and select the "Update Driver" option.

5. Select the "Browse My Computer for Driver Software" option and then select "Let me pick from a list of available drivers on my computer".

6. If a list of categories is provided, select "all categories" and click next.

7. Select the "Have Disk" option and then browse to the location of the .inf file downloaded in Step 1.

8. Install the driver following instructions on screen.

6.0 System and Module Initialization

MAQ®20 functionalities provided by the C API are imported through the following include statements:

```
1. // MAQ20 C API header files
2. #include "MAQ20_System.h
3. #include "MAQ20_Module.h
```

MAQ20_System.h contains function prototypes used for initializing and interfacing with a MAQ®20 system as a whole. Initialization of the system occurs by way of a call to the `init_maq20` function in this file. When successful, this function returns a pointer to an initialized `_maq20` structure for further usage.

```
1. // Function prototype for initializing a MAQ20 system
2. maq20* init_maq20(char* protocol, char* ip_address, char* port, int baud_rate, char*
3.                   parity, char* com_port, maq20_com* com, int timeout, int slave_id);
```

Individual modules in a MAQ®20 system can then be initialized in a similar manner with a call to the `init_modules` function in MAQ20_Modules.h, which returns a pointer to the initialized `_maq20_module` structure.

```
1. // Function prototype for initialization of a single MAQ20 module
2. maq20_module* init_module(maq20_com* com, int registration_number);
```

Example initialization code for a MAQ®20 system composed of two modules:

```
 1. // Connect to MAQ20 System through a TCP connection
 2. maq20* system;
 3. system = init_maq20("TCP", "192.168.128.100", "502", 9600, "None",
 4.                     "COM6", NULL, DEFAULT_TIMEOUT, 16);
 5. if (system == NULL)
 6.   return -1;
 7.
 8. // Print module information for the MAQ20 System
 9. for (int i = 0; i < 3; ++i) {
10.   maq20_module* module = init_module(system->com, i);
11.   if (module == NULL)
12.         continue;
13.   printf("%s\n", module_information(module));
14.   close_module(module);
15. }
```

Console Output:

```
 1. MAQ20-COM4
 2. Registration Number: 0
 3. Serial Number -----: S0113207-05
 4. Date Code ---------: D0723
 5. Firmware Version --: F1.81
 6.
 7. MAQ20-VDN
 8. Registration Number: 1
 9. Serial Number -----: S0098692-19
10. Date Code ---------: D1223
11. Firmware Version --: F1.70
12. Input Channels ----: 8
13. Output Channels ---: 0
14.
15. MAQ20-VO
16. Registration Number: 2
17. Serial Number -----: S0120816-01
18. Date Code ---------: D0523
19. Firmware Version --: F1.70
20. Input Channels ----: 0
21. Output Channels ---: 8
```

In the example code shown above, a system instance was created and initialized upon successfully connecting to a MAQ®20 system located at IP address 192.168.128.100. Module instances were then created and initialized in memory with information for each module printed out through a call to the `module_information` function.

While locating and initializing of modules in a system can be done in this manner, the C API allows for module initialization to also occur by searching for a module's name or unique serial number. The ability to search by module serial number can be useful in cases where multiple modules of the same type exist within a given MAQ®20 System. The function prototype is as follows:

```
1. // Function prototypes for locating a module based on its name or unique serial number
2. maq20_module* find_module(maq20* system, char* name_or_sn);
```

Example calls to this function can be found in the example code located in Section 10.0 of this manual.

7.0 Direct Reading and Writing of Registers

Read and write requests for single and multiple registers can be issued at the system-level with absolute addressing, or at the module-level with relative addressing. If desired, a MAQ®20 system can be controlled and interacted with entirely these low-level methods.

At the system-level the following functions from MAQ20_System.h are used for reading and writing to absolute register locations specified by the integer type address parameter:

```
1. // Function prototypes for single and multiple read absolute register(s)
2. int read_register(maq20* system, INT16S* dest, int address);
3. int read_registers(maq20* system, INT16S* dest, int address, int num_regs);
4.
5. // Function prototypes for single and multiple write absolute register(s)
6. int write_register(maq20* system, int address, INT16S value);
7. int write_registers(maq20* system, int address, int num_regs, INT16S values[]);
```

Relative reading and writing functions defined in MAQ20_Module.h occur at the module-level and specify the integer parameter address as an offset from the module's starting address:

```
 1.  // Function prototypes for single and multiple read relative register(s)
 2.  int read_register_relative(maq20_module* module, INT16S* dest, int address);
 3.  int read_registers_relative(maq20_module* module, INT16S* dest, int address,
 5.                              int num_regs);
 6.
 7.  // Function prototypes for single and multiple write relative register(s)
 8.  int write_register_relative(maq20_module* module, int address, INT16S value);
 9.  int write_registers_relative(maq20_module* module, int address, int num_regs,
10.                               INT16S values[]);
```

8.0 Reading and Writing Channel Data

While reading and writing of individual or multiple channels of data can be done using the methods outlined in Section 7.0, the MAQ®20 API provides functionalities to make this process easier for module channel data. MAQ20_Module.h contains function prototypes that allow for single or multiple channels to be read solely by specifying the module and channel(s) to be operated on. Function implementations allow for units to be specified in raw counts or engineering units.

The following functions provide read and write functionalities at the module-level in counts:

```
// Function prototypes for single and multiple read channel(s) in counts
int read_channel_data_counts(maq20_module* module, int channel, INT32S* dest);
int read_data_counts(maq20_module* module, int start_channel, int num_channels,
                     INT32S dest[]);

// Function prototypes for single and multiple write channel(s) in counts
int write_channel_data_counts(maq20_module* module, int channel, INT32S data);
int write_data_counts(maq20_module* module, int start_channel, int num_channels,
                      INT32S data_set[]);
```

Function implementations that allow for specifying values in engineering units handle the conversion of engineering units to counts for write operations and counts to engineering units when reading:

```
// Function prototypes for single and multiple read channel(s) in engineering units
int read_channel_data_eng_values(maq20_module* module, int channel, double* dest);
int read_data_eng_values(maq20_module* module, int start_channel, int num_channels,
                         double dest[]);

// Function prototypes for single and multiple write channel(s) in engineering units
int write_channel_data_eng_values(maq20_module* module, int channel, double data);
int write_data_eng_values(maq20_module* module, int start_channel, int num_channels,
                          double data_set[]);
```

9.0 Freeing Memory

The MAQ®20 C API provides functionalities for freeing memory at system and module-levels. During program execution, if no further interactions with a previously initialized module are required, the API provides a functionality for freeing the memory associated it by a call to `close_module`:

```
// Free memory associated with module instance
void close_module(maq20_module* module);
```

At the end of program execution memory associated with the entire MAQ®20 system should be freed with a call to `close_system`:

```
// Free memory associated with system instance
void close_system(maq20* system);
```

When memory is freed at the system-level there is no prior need to free memory at the module-level as it occurs automatically through the system-level call.

10.0 Example Code

The following code implementation of Example.c from Example.sln demonstrates usage of the functions described in Sections 6-9 of this manual. The MAQ®20 system used for this example was composed of a MAQ20-COM4 module with a MAQ20-VDN module registered in slot 1 and a MAQ20-VO module registered in slot 2. All channels of the MAQ20-VO module were tied to

equivelent channel numbers of the MAQ20-VDN module. The code demonstrates initializing the MAQ®20 system, finding specific modules by name, absolute and relative register operations, module configuration, and reading of module channels:

```c
#include "MAQ20_System.h"
#include "MAQ20_Module.h"
#include "constants.h"
#include "Example.h"
#include <stdio.h>

int main() {

    // Connect to MAQ20 System through a TCP connection
    maq20* system;
    system = init_maq20("TCP", "192.168.128.100", "502", 9600, "None",
                        "COM6", NULL, DEFAULT_TIMEOUT, 16);
    if (system == NULL)
      return -1;

    // Print module information for the MAQ20 System
    for (int i = 0; i < 3; ++i) {
        maq20_module* module = init_module(system->com, i);
        if (module == NULL)
            continue;
        printf("%s\n", module_information(module));
        close_module(module);
    }

    // Locate modules by name and create instances of them
    maq20_module* vdn1 = find_module(system, "MAQ20-VDN");
    maq20_module* vo1 = find_module(system, "MAQ20-VO");

    /**
     * If range information is incorrect, set channel ranges for both
     * modules to +/-10V and start over with correctly initialized ranges
     */
    INT16S vdnCurRange[8] = {9, 9, 9, 9, 9, 9, 9, 9};
    INT16S voCurRange[8] = {9, 9, 9, 9, 9, 9, 9, 9};
    INT16S vdnRangeConfig[8] = {3, 3, 3, 3, 3, 3, 3, 3};
    INT16S voRangeConfig[8] = {0, 0, 0, 0, 0, 0, 0, 0};

    /**
     * Read current module range settings
     * Absolute (system-level) addressing used for reading current MAQ20-VDN ranges
     * Relative (module-level) addressing used for reading current MAQ20-VO ranges
     */
    read_registers(system, vdnCurRange, 2100, 8);
    read_registers_relative(vo1, voCurRange, 100, 8);

    // Check if current range matches desired
    for (int i = 0; i < 8; ++i) {
        if ((vdnCurRange[i] != vdnRangeConfig[i])||(voCurRange[i] != voRangeConfig[i])){
            /**
             * Absolute (system-level) addressing used for MAQ20-VDN range assignment
             * Relative (module-level) addressing used for MAQ20-VO range assignment
             */
            write_registers(system, 2100, 8, vdnRangeConfig);
            write_registers_relative(vo1, 100, 8, voRangeConfig);

            // Close system and reinitialize system and modules with corrected ranges
            close_maq20(system);
            system = init_maq20("TCP", "192.168.128.100", "502", 9600, "None",
                                "COM6", NULL, DEFAULT_TIMEOUT, 16);
            vdn1 = find_module(system, "MAQ20-VDN");
```

```
61.             vo1 = find_module(system, “MAQ20-VO”);
62.         {
63.     {
64.
65.     // Set voltage of CH0 of MAQ20-VO module to 2243 counts (+1V)
66.     write_channel_data_counts(vo1, 0, 2243);
67.
68.     // Set voltage of CH1-CH6 of MAQ20-VO module to +2V, +3V, ..., +7V in counts
69.     INT32S voCh1To6Cnts[6] = {2438, 2633, 2828, 3023, 3218, 3413};
70.     write_data_counts(vo1, 1, 7, voCh1To6Cnts);
71.
72.     // Set voltage of CH7 of MAQ20-VO module to +8.75V (3754 Counts) in ENG units
73.     write_channel_data_eng_values(vo1, 7, (double)8.75);
74.
75.     // Read voltage of CH0 of MAQ20-VO module in counts and print result
76.     INT32S voCh0Cnts = 0;
77.     read_channel_data_counts(vo1, 0, &voCh0Cnts);
78.     printf("\nMAQ20-VO CH0 (COUNTS): %d\n", voCh0Cnts);
79.
80.     // Read voltage of CH1-CH7 of MAQ20-VO module in counts and print result
81.     INT32S voCh1to7CountVals[7] = {0, 0, 0, 0, 0, 0, 0};
82.     read_data_counts(vo1, 1, 7, voCh1to7CountVals);
83.     printf("MAQ20-VO CH1-CH7 (COUNTS): ");
84.     print_counts(voCh1to7CountVals, 7);
85.
86.     // Read voltage of CHO of MAQ20-VDN module in ENG units and print result
87.     double vdnCh0EngVal = 0;
88.     read_channel_data_eng_values(vdn1, 0, &vdnCh0EngVal);
89.     printf("\nMAQ20-VDN CH0 (ENG): %.4f", vdnCh0EngVal);
90.
91.     // Read voltage of CH1-7 of MAQ20-VDN module in ENG units and print result
92.     double vdnCh1to7EngVals[7] = {0,0,0,0,0,0,0};
93.     read_data_eng_values(vdn1, 1, 7, vdnCh1to7EngVals);
94.     printf("\nMAQ20-VDN CH1-CH7 (ENG): ");
95.     print_eng(vdnCh1to7EngVals, 7);
96.
97.     // Free Memory
98.     close_maq20(system);
99.
100.    return 0;
101.
102. }
```

Console Output:

```
 1. MAQ20-COM4
 2. Registration Number: 0
 3. Serial Number -----: S0113207-05
 4. Date Code ---------: D0723
 5. Firmware Version --: F1.81
 6.
 7. MAQ20-VDN
 8. Registration Number: 1
 9. Serial Number -----: S0098692-19
10. Date Code ---------: D1223
11. Firmware Version --: F1.70
12. Input Channels ----: 8
13. Output Channels ---: 0
14.
15. MAQ20-VO
16. Registration Number: 2
17. Serial Number -----: S0120816-01
18. Date Code ---------: D0523
19. Firmware Version --: F1.70
```

```
20. Input Channels ----: 0
21. Output Channels ---: 8
22.
23.
24. MAQ20-VO CH0 (COUNTS): 2243
25. MAQ20-VO CH1-CH7 (COUNTS) 2438 2633 2828 2023 3218 3413 3754
26.
27. MAQ20-VDN CH0 (ENG): 0.9985V
28. MAQ20-VDN CH1-CH7 (ENG): 1.9957V 2.9970V 3.9965V 4.9982V 5.9985V 6.9991V 8.7497V
```

DATAFORTH WARRANTY

To view the current Dataforth Corporation Warranty, please click on the link below for the Dataforth Standard Terms and Conditions of Sale Applying to Products Sold by Dataforth Corporation. The Warranty in its entirety is Section 3. Please check this link periodically for updates.

https://www.dataforth.com/terms-and-conditions-sale

APPLICATION SUPPORT

Dataforth provides timely, high-quality product support. Send an email to support@dataforth.com or call **+1-800-444-7644 TOLL-FREE.**

Returns/Repair Policy

All warranty and repair requests should be directed to the Dataforth Customer Service Department at +1-520-741-1404 or **+1-800-444-7644 TOLL-FREE.**

If a product return is required, submit a Return Material Authorization (RMA) request by visiting https://www.dataforth.com.

MAQ®20 C API User Manual
MA1070 Rev. A – March 2024

ISO9001:2015-Registered QMS