User's manual
Development Kit for LabView

QUALIFLOW Montpellier
350, rue A. Nobel
BP7- 34935 MONTPELLIER
CEDEX 9
France
tel: +33 4 67 99 47 47
fax: +33 4 67 99 47 48

QUALIFLOW Technology Center
44862 Osgood Road
Fremont, CA-94539
CALIFORNIA - USA
tel: +1 510 440 93 74
fax: +1 510 440 93 75

QUALIFLOW NRT
10 Block-17 Lot, Namdong Ind.
CLX. #623-16n Namchon-Dong, Namdong-Ku
KOREA
tel: +82 (0)2 3401 6491
fax: +82 (0)2 3401 6493
<table>
<thead>
<tr>
<th>Identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
</tr>
<tr>
<td>Document name</td>
</tr>
<tr>
<td>File name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
</tr>
<tr>
<td>Visa</td>
</tr>
<tr>
<td>Albin Diranze</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
</tr>
<tr>
<td>Albin Diranze</td>
</tr>
<tr>
<td>Section</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>1.</td>
</tr>
<tr>
<td>2.</td>
</tr>
<tr>
<td>3.</td>
</tr>
<tr>
<td>4.</td>
</tr>
<tr>
<td>5.</td>
</tr>
<tr>
<td>6.</td>
</tr>
<tr>
<td>6.1</td>
</tr>
<tr>
<td>6.2</td>
</tr>
<tr>
<td>6.2.1</td>
</tr>
<tr>
<td>6.2.2</td>
</tr>
<tr>
<td>6.2.3</td>
</tr>
<tr>
<td>6.2.4</td>
</tr>
<tr>
<td>6.2.5</td>
</tr>
<tr>
<td>6.2.6</td>
</tr>
<tr>
<td>6.2.7</td>
</tr>
<tr>
<td>6.2.8</td>
</tr>
<tr>
<td>6.2.9</td>
</tr>
<tr>
<td>6.2.10</td>
</tr>
<tr>
<td>6.2.11</td>
</tr>
<tr>
<td>7.</td>
</tr>
<tr>
<td>7.1</td>
</tr>
<tr>
<td>7.2</td>
</tr>
<tr>
<td>7.3</td>
</tr>
<tr>
<td>7.4</td>
</tr>
<tr>
<td>7.5</td>
</tr>
<tr>
<td>7.6</td>
</tr>
<tr>
<td>7.7</td>
</tr>
<tr>
<td>7.8</td>
</tr>
<tr>
<td>7.9</td>
</tr>
<tr>
<td>7.10</td>
</tr>
<tr>
<td>7.11</td>
</tr>
</tbody>
</table>
1. Description

This document describes the different LabView VI’s utilized to supervise and control Qualiflow Digital Mass Flow Controllers. These VI’s interface the Modbus Dynamic Link Library to communicate with MFC’s. “MFC Monitor” application is given as example.

2. Scope

The MFC Development Kit for LabView is based on the explanation of a simple application example. Users are supposed to know LabView.
“MFC Monitor” Labview program has been developed with LabView 6.0.
MFC Firmware version 7.33 or higher is necessary to run this program.

3. References

The following documents were used as basis for constructing this present document. They represent also an interesting source of information.

Ref 1  [QUALIFLOW] SP 0003 Digital MFC Memory Layout V06

Ref 2  [QUALIFLOW] SP 0006 MFC Application Program Interface V01


4. Glossary

DLL : Dynamic Link Library
VI : Virtual Instrument
MFC : Mass Flow Controller
5. Program architecture

The Modbus protocol is encapsulated in a Dynamic Link Library (modbus.dll). This DLL accesses the different MFC's through RS-232 or RS-485 serial link. Note that only one MFC can be addressed with RS-232. A full description of the DLL is given in Ref 2

The example program is based on sequences. Each sequence carries out a special action and guarantees the sequential access to the serial link. Although other architectures may also be adequate, this is a comprehensive way to do for those developers who are not experts in LabView.

Start

Variables initialization

DLL initialization

Open communication port

Read configuration data (identifiers, serial number … )

Main loop

Read dynamic data (actual flow, alarm … )

Close communication port

Free DLL

Exit
6. Main VI

6.1 Front panel

![mfc_monitor.vi front panel](image)

This is the main window of the application “MFC Monitor”.
Grey entry fields are read-only.
“MFC Monitor” application is constructed in sequences. Detail of these sequences is given bellow.

Inter frame
The inter frame entry field holds the time (in ms) the application will wait between two Modbus transactions.
This allows MFC’s to synchronize in case of data collision. Inter frame time can be adjusted.
Typical values are:
- 150 ms for 2400 Bauds
- 50 ms for 9600 Bauds
6.2 mfc_monitor.vi

6.2.1 Memory initialization

Description
This sequence is to make sure the correct amount of memory is allocated to each variable before the DLL is called. For doing so, local variables of indicators are created and filled with blanks.
6.2.2 DLL initialization

Description
This sequence initializes the DLL.
“InitDLL” VI is used.
See chapter 7.1 for more detail on “InitDLL” VI.

If the call fails, an error message is raised up and the program terminates.
6.2.3 Open com port

Description
Open COM port by calling “OpenPort” VI. Baud rate is specified by “Baud rate” indicator, Port number by “Port number” indicator. A false response raises an error message and stops the program. Note that others configuration parameters such as Parity and Data length are not accessible. Default parameters:
- Baud rate: 2400
- Start bits: 1
- Parity: none
- Data length: 8
- Stop bits: 1

At this step, the communication port is open and configured. We can now start reading information from or writing commands to MFC’s. See chapter 7.2 for more detail on “OpenPort” VI.

Figure 4: Open com port
6.2.4 Read software version

Description
In this sequence, the software version is read. A DLL call function is used.

Function prototype
?? unsigned char MFC_READ(long arg1, unsigned short int arg2, CStr arg3, long arg4);

Arguments
?? arg1 Modbus address Command “MFC address”
?? arg2 Modbus data base address 547 (0x0223)
?? arg3 Pointer to data destination Indicator “Soft version” (1)
?? arg4 Type of pointer 5 (char[4])

Return value
?? TRUE : data is copied to destination
?? FALSE : an error message is raised and program is stopped

(1) Memory is allocated in sequence 0 of 10
6.2.5  Read manufacturer identifier

Description
In this sequence, the manufacturer identifier is read. A DLL call function is used.

Function prototype
```c
unsigned char MFC_READ(long arg1, unsigned short int arg2, CStr arg3, long arg4);
```

Arguments
- `arg1`: Modbus address
- `arg2`: Modbus data base address
- `arg3`: Pointer to data destination
- `arg4`: Type of pointer

Return value
- `TRUE`: data is copied to destination
- `FALSE`: an error message is raised and program is stopped

(1) Memory is allocated in sequence 0 of 10
6.2.6 Read serial number

Description
In this sequence, the serial number is read.
A DLL call function is used.

Function prototype

```c
unsigned char MFC_READ(long arg1, unsigned short int arg2, CStr arg3, long arg4);
```

Arguments

- `arg1`: Modbus address  Command “MFC address”
- `arg2`: Modbus data base address  527 (0x020F)
- `arg3`: Pointer to data destination  Indicator “Serial number” (1)
- `arg4`: Type of pointer  6 (char[10])

Return value

- TRUE : data is copied to destination
- FALSE : an error message is raised and program is stopped

(1) Memory is allocated in sequence 0 of 10
6.2.7 Read product identifier

Description
In this sequence, the product identifier is read.
A DLL call function is used.

Function prototype

```c
unsigned char MFC_READ(long arg1, unsigned short int arg2, CStr arg3, long arg4);
```

Arguments

- `arg1`: Modbus address  
  Command “MFC address”
- `arg2`: Modbus data base address  
  517 (0x0205)
- `arg3`: Pointer to data destination  
  Indicator “Product ID” (1)
- `arg4`: Type of pointer  
  6 ( char[10] )

Return value

- `TRUE`: data is copied to destination
- `FALSE`: an error message is raised and program is stopped

(1) Memory is allocated in sequence 0 of 10
6.2.8 Read valve type

Description
In this sequence, the valve type is read.
The call to the DLL function is embedded in “ReadU8” VI which returns:
?? 1 for normally closed
?? 0 for normally open

If the call fails, an error message is raised and the program terminates.
See chapter 7.6 for more detail on “ReadU8” VI.
6.2.9 Main loop sequence

The program’s main loop is also organized in sequences.
- Manage the “Write setpoint” command
- Manage the “Clear alarm” command
- Manage the “Write override” command
- Read actual values from the MFC

Main loop terminates when “Stop” button is pressed.

Figure 10: Main loop
6.2.9.1 Write setpoint

**Description**
In this sub-sequence, the “Write setpoint” button is checked. If pressed, the setpoint is converted into binary value and sent to the MFC. “WriteU16” VI is used.
See chapter 7.8 for more detail on “WriteU16” VI.

**Equivalent flow chart**

---

To be defined V00
6.2.9.2 Clear alarm

Description
In this sub-sequence, the “Clear alarm” button is checked. If pressed, the command to clear the alarm is sent to the MFC. “WriteCoil” VI is used. See chapter 7.7 for more detail on “WriteCoil” VI.

Equivalent flow chart

![Flowchart for Clear alarm]
6.2.9.3 Write override

**Figure 13 : Write override**

**Description**
In this sub-sequence, the “Write override” button is checked. If pressed, the override command byte is sent to the MFC. “WriteU8” VI is used.
See chapter 7.9 for more detail on “WriteU8” VI.

**Equivalent flow chart**

```
<table>
<thead>
<tr>
<th>NO</th>
<th>Write override button pressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>Write byte</td>
</tr>
<tr>
<td></td>
<td>Write success</td>
</tr>
<tr>
<td></td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Inter frame wait</td>
</tr>
<tr>
<td></td>
<td>Raise error message</td>
</tr>
<tr>
<td></td>
<td>Go to next step</td>
</tr>
</tbody>
</table>
```
6.2.9.4 Read data

Description
This sub-sequence reads data from MFC.
“ReadBatch” VI is used.
See chapter 7.3 for more detail on “ReadBatch” VI.

Equivalent flow chart

- Read data
  - Read data
    - Read data
      - success
        - NO
        - YES
          - Update indicators
            - Inter frame wait
              - Raise error message
            - Go to next step
6.2.10 Close com port

Description
This sequence closes the communication port.
“ClosePort” VI is used.
See chapter 7.1 for more detail on “ClosePort” VI.

If the call fails, an error message is raised and the program terminates.
6.2.11 Release DLL

Description
This sequence releases the Dynamic Link Library. “FreeDLL” VI is used. See chapter 7.11 for more detail on “FreeDLL” VI.

If the call fails, an error message is raised and the program terminates.
7. Sub-VI's

All sub-VI’s call the Modbus DLL.

7.1 InitDLL.vi

![Diagram of InitDLL.vi]

**Description**

DLL initialization.

**C Function prototype**

```c
unsigned char MFC_InitDLL(void);
```

**Arguments**

- **none**

**Return value**

- **TRUE**: DLL initialized correctly
- **FALSE**: DLL initialization failed

7.2 OpenPort.vi

![Diagram of OpenPort.vi]

**Description**

Open a communication port.

**C Function prototype**

```c
unsigned char MFC_OpenPort(long arg1, long arg2);
```

**Arguments**

- **arg1**: Port number (1 for COM1; 2 for COM2 ...)
- **arg2**: Baud rate (2400 or 9600)

**Return value**

- **TRUE**: Com port opened and initialized correctly
- **FALSE**: Com port initialization failed
7.3 ReadBatch.vi

Description
Read a data cluster. Data read with this function is stored in a string. The returned string is scanned and data is broken down into simple variables.

C Function prototype
```c
unsigned char MFC_READ(long arg1, unsigned short int arg2, CStr arg3, long arg4);
```

Arguments
- `arg1`: Modbus address Command “MFC address”
- `arg2`: Modbus data base address 2 (0x0002)
- `arg3`: Pointer to data destination Indicator “String” (1)
- `arg4`: Type of pointer 11

Return value
- **TRUE**: Read data succeeded and data is copied to destination
- **FALSE**: Read data failed

(1) Note that “String” is initialized with the correct amount of memory before the call.
7.4 Readout.vi

**Description**
Read the actual readout. Unit is percent of full scale.

**Function prototype**
```
unsigned char MFC_READ(long arg1, unsigned short int arg2, float *arg3, long arg4);
```

**Arguments**
- **arg1**: Modbus address  Command “MFC address”
- **arg2**: Modbus data base address  184 (0x00B8)
- **arg3**: Pointer to data destination  IEEE-754 32 bits Float
- **arg4**: Type of pointer  3

**Return value**
- **TRUE**: Read data succeeded and data is copied to destination
- **FALSE**: Read data failed

7.5 ReadU16.vi

**Description**
Read an unsigned 16 bits variable.

**C Function prototype**
```
unsigned char MFC_READ(long arg1, unsigned short int arg2, unsigned short int *arg3, long arg4);
```

**Arguments**
- **arg1**: Modbus address  Command “MFC address”
- **arg2**: Modbus data base address  Command “Variable address”
- **arg3**: Pointer to data destination  unsigned 16 bits
- **arg4**: Type of pointer  1

**Return value**
- **TRUE**: Read data succeeded and data is copied to destination
- **FALSE**: Read data failed
7.6 ReadU8.vi

Description
Read an unsigned 8 bits variable.

C Function prototype
```c
unsigned char MFC_READ(long arg1, unsigned short int arg2, unsigned char *arg3, long arg4);
```

Arguments
- `arg1`: Modbus address
  Command “MFC address”
- `arg2`: Modbus data base address
  Command “Variable address”
- `arg3`: Pointer to data destination
  unsigned 8 bits
- `arg4`: Type of pointer
  0

Return value
- TRUE: Read data succeeded and data is copied to destination
- FALSE: Read data failed
7.7 **WriteCoil.vi**

**Figure 23 : WriteCoil.vi**

Description
Write a coil variable.

**C Function prototype**

```c
unsigned char MFC_WRITE(long arg1, unsigned short int arg2, unsigned char *arg3, long arg4);
```

**Arguments**

- **arg1** Modbus address Command “MFC address”
- **arg2** Modbus data base address Command “Variable address”
- **arg3** Pointer to data source unsigned 8 bits
- **arg4** Type of pointer 2

**Return value**

- **TRUE** : Write data succeeded
- **FALSE** : Write data failed

---

7.8 **WriteU16.vi**

**Figure 24 : WriteU16.vi**

Description
Write an unsigned 16 bits variable.

**C Function prototype**

```c
unsigned char MFC_WRITE(long arg1, unsigned short int arg2, unsigned short int *arg3, long arg4);
```

**Arguments**

- **arg1** Modbus address Command “MFC address”
7.9 WriteU8.vi

Description
Write an unsigned 8 bits variable. Modbus does not allow transmission of a single byte. Therefore, this VI used the same parameters as “WriteU16” VI. Data byte must be transmitted in the most significant byte of the word.

C Function prototype
```c
unsigned char MFC_WRITE(long arg1, unsigned short int arg2, unsigned short int *arg3, long arg4);
```

Arguments
- **arg1**: Modbus address Command “MFC address”
- **arg2**: Modbus data base address Command “Variable address”
- **arg3**: Pointer to data source unsigned 16 bits
- **arg4**: Type of pointer 1

Return value
- **TRUE**: Write data succeeded
- **FALSE**: Write data failed
7.10 ClosePort.vi

Figure 26: ClosePort.vi

Description
Close a communication port.

Function prototype
```c
unsigned char MFC_ClosePort(long arg1);
```

Arguments
```c
arg1 : Port number (1 for COM1; 2 for COM2 …)
```

Return value
```c
TRUE : Com port closed correctly
FALSE : Com port close failed
```

7.11 FreeDLL.vi

Figure 27: FreeDLL.vi

Description
Free and release Modbus DLL.

Function prototype
```c
unsigned char MFC_FreeDLL(void);
```

Arguments
```c
none
```

Return value
```c
TRUE : DLL released correctly
FALSE : DLL release failed
```