RHE4x Desktop Reference Data Logging

GET FLOW MEASURED

Rheonik Messtechnik GmbH
Rudolf-Diesel-Straße 5
D-85235 Odelzhausen
Germany

Tel. + 49 (0)8134 9341-0
info@rheonik.com

LEARN
MORE

Document No.: 8.2.1.15
Version 1.00
October 2020

RHE4X Transmitter

Addendum Desktop Reference Data Logging

RHEONIK.

Table of Contents

Baseline reference:

Version 0.10 of this document reflects the properties of the RHE4X firmware version 1.68 and later.

Version 0.20 of this document reflects the properties of the RHE4X firmware version 2.00 and later. The corresponding RHEComPro version is 3.0.6.

1 Introduction

Some types of transmitters of the RHE4X series are able to log its status, the flow and temperature measurements as well as the totalizers and other parameters onto a solid state memory inside the RHE transmitter for quite a while. The recorded data is intended to provide customers with additional information for the optimization of their ongoing processes and the ability to analyze potential failures in their installation.

Since the handling of the Data Logging Feature and its related data is rather complex its description is not included in the “RHE4X Desktop Reference Manual”, Document Number 8.2.1.14, but it was decided to create a separate document for it.

This document describes the Data Logging Feature on several detail levels from user handling up to the level needed to write own software handling the potential huge amount of data. This does not seem necessary since the RHEComPro program is able to perform standard operations with the recorded data like writing a CSV file from a selected range of recorded data.

This document contains three major sections:

- Principles of the Data Logging Feature
- User Interface to the Data Logging Feature
- Detailed Description of the Software Interfaces

The first section describes the principles of the Data Logging Feature and should be read by user who intend to employ this feature. Not all technical details mentioned in there need to be understood because all related and derived facts are summarized in clear statements.

The second section describes the handling of the Data Logging with the help of the RHEComPro program. This is the easiest way to handle the potentially huge amount of data generated by the Data Logging Feature and therefore, this section should be read by users who want to handle the recorded data on a PC and for whom the functionality of the RHEComPro program is sufficient.

The third section describes the software interfaces of the Data Logging Feature in detail and is recommended only to users who need to write their own software to handle the recorded data. This section is complemented by the appendices which contain the data structures needed for the implementation ready to be used for the C and C# programming languages.

2 Principles of the Data Logging Feature

Transmitters of the RHE4X series which support the Data Logging Feature contain a soldered-in solid-state memory in form of a serial NAND Flash. This NAND Flash is at least 128MB in size.

Almost all transmitter-internal status and measurement data is assembled into a data record which may be periodically written to the flash memory. The time interval for writes of such data records to the flash memory can be selected by the user in the range of 1 to 600 seconds. The size of the data record amounts to 256 Bytes. This means that the flash memory may contain data of at least six days when the data logging interval is set to one second. This time span increases respectively when the data logging interval is increased.

The data logging interval and the start/stop state of the logging is controlled by following Modbus holding registers of the “Generic” holding register group:

Modbus Address	Register Name	Description
0x60D2	RecordingRequest	(RHE4X) **Data Logging (Recording Request):** Control of the Data Logging Feature. 0 Stops the logging, 1 starts the logging. A request changes becomes effective immediately and remains valid even through a power cycle or a system reset. Default is 0.
0x60D4	RecordingInterval	(RHE4X) **Data Logging (Recording) Interval:** Data Logging interval in seconds. The interval between two data records may be chosen from 1 to 600 seconds. The default is 1.

Table 1: Modbus Input Registers for the Administration of the Loggings

The flash memory is written in a circular manner. The logging starts at the "bottom" (lowest address) of the flash memory and is continued to the "top" of the memory (higher addresses). When the topmost addresses are written, the writing continues at the bottom and the data stored there previously is lost. This scheme is maintained even when the system is reset or a power cycle occurred. Thus, it is made certain that only the oldest data is overwritten when the data logging is restarted.

Each record is identified by a 32-bit "Record Id" number. This number is incremented whenever a new record is written. These numbers are unique over normal deployment durations of RHE transmitters. At the minimum data logging interval of one second it would last over 130 years until this identification would wrap from its maximum value 4294967295 to 0. Even if this would happen the record ids in the stored records would remain unique.

Aside from its own record id a record contains additional administrative data, e.g. another record id which refers to the first record that started the current logging sequence. This "Reset Record Id" is identical in all records of the same sequence. Thus, it may be used to search for the end of the previous data logging sequence. When a record at the end of the previous logging sequence can be read its "Reset Record Id" will refer to the start of the previous logging sequence which then can be used to search backwards for older sequences.

Any search for older sequences must always be aware of the possibility that this sequence has been overwritten by new data records partly or totally. Since the reading of data records can be performed while the logging is active new data records the transmitters offers following information in Modbus input record registers:

Modbus Address	Register Name	Description
0x4034	RecordingMinId	(RHE4X) **Data Logging - Minimum Record Id:** Current minimum identification in the data logging sequence.
0x4036	RecordingMaxId	(RHE4X) **Data Logging - Maximum Record Id:** Current maximum identification in the data logging sequence.
0x4038	RecordingLastResetId	(RHE4X) **Data Logging - Record Id at Last Reset:** Identification in the data logging sequence which belongs to the start of the data logging after the last system reset or a restart of the recording via RecordingRequest 0x60D2.
0x403A	RecordingResetTime	(RHE4X) **Data Logging - Recording Time at Last Reset:** Time stamp in the record ID which is used in the record belonging to the last reset identification. The time stamp is expressed in seconds since midnight 1980-01-01. The correctness of the time stamp depends on the correctness of the system time, see registers 0x60B0 to 0x60BE.

Modbus Address	Register Name	Description
0x403C	RecordingMaxTime	(RHE4X) **Data Logging - Recording Maximum Time:** Current maximum time stamp in the record ID which is used in the record belonging to the maximum identification. The time stamp is expressed in seconds since midnight 1980-01-01. The correctness of the time stamp depends on the correctness of the system time, see registers 0x60B0 to 0x60BE.
0x403E	RecordingStatus	(RHE4X) **Data Logging - Recording Status:** Current status of the data logging feature. In the least signification byte following values are found: 0: Logging available and stopped. 1: Logging running. 2: Flash Erase in Progress. 3: Fatal Error occurred. 4: Logging not available. In the second byte an additional error status is found when a fatal error is signaled.

Table 2: Modbus Input Registers for the Administration of the Loggings

The registers RecordingMinId and RecordingMaxId contain the current minimum and maximum record ids present in the flash memory. This information is updated whenever new records are written or older records are erased and should be used to determine whether a record belonging to a record id is accessible or not.

The flash memory must be erased before new data can be written. The minimum unit that can be erased in the NAND flash is a block usually of 128kB in size which corresponds to 512 data records. When a block is erased the first record in a block does not contain measurement data but a copy of important setup parameters.

This is done because the measurement data sometimes can only be interpreted correctly when the corresponding setup parameters are known. Thus, all records whose record ids are divisible by 512 without remainder should contain setup parameters. Such parameter records contain a special flag setting that they are not misinterpreted as a record of measurement data. When a logging is started the very first record also will contain setup data. Therefore, the setup flag should be checked before the record is interpreted.

When the RHEComPro program generates a CSV file and it could read one of the setup records in the logging sequence one set of setup data is written in the first data line behind the columns of measurement data. This allows for a linked operation on the measurement data based on the setup data.

Another administrative data in the records is a time stamp “time_since_reset” that is based on millisecond units since the last reset of the software. This stamp is unique for about 50 days. It can be kept unique in Excel sheets beyond this time span by adding 4294967296 to the reset of the data column whenever it wraps from 4294967295 to 0. It is recommended to use this time stamp as time base for plots because the time/date related time stamp described below may be modified by user or network activities during the ongoing logging.

Another time stamp in the records is “time_stamp”, an unsigned 32-bit numbers which contains the number of seconds since midnight 1980-01-01. This time stamp format will result in unique numbers until the year 2116 and may be converted to other time stamp formats, e.g. to the format used by Excel which is based on midnight 1900-01-01 by adding the number 2524694400. It is recommended to perform this addition on 64-bit data items in order to avoid problems after the year 2036. These numbers can be converted to Excel date/time information by dividing the numbers by 86400 (seconds in a day) and setting the format of the cell to a user defined format which displays the day as well as the time information. When RHEComPro generates a CSV file it already writes out time stamps fit for Excel in the required floating point format the integer part of which is based on the number of days since 1900-01-01.

Since this time stamp may be modified during the ongoing logging any modification is marked by a special flag in the “flags” field of the record. The “flags” field is a 16-bit bit set which marks important events which may affect the recorded data or its interpretation. The bits have following meaning:

Bit	**Mask**	**Meaning**
0:	0x0001	A system reset or power cycle occurred since the previous data logging sequence.
1:	0x0002	Data logging has been stopped by a user command. The flags marks the last record of a logging sequence.
2:	0x0004	Data logging has been started from the stopped state by a user command. The flags marks the first record of a new logging sequence.
3:	0x0008	The system date/time has been changed and may be reflected by a jump forwards or backwards in the “time_stamp” field of the records.
4:	0x0010	The totalizers have been reset via user commands. They will continue with a start value of 0.
5:	0x0020	The totalizers are currently stopped. All records during the stopped state of the totalizers will contain this flag. No (normal) totalizer increment will occur.
6:	0x0040	A system reset has been commanded. This flags indicates the last record of a command sequence and will be followed by a system shutdown.
7:	0x0080	A Zeroing procedure has been initiated by the user. After completion of the Zeroing the mass flow calculation will continue with a different ZeroPoint. The current ZeroPoint is part of the measurement data record.
15:	0x8000	Records contains setup data instead of measurement data.

Table 3: Bit definitions in the “Flags” field

The records of 256 bytes in size are bundled together in sets of 8 records to form a 2048-byte “page” in the NAND flash. A page is a unit in the flash memory which can be written in one operation. Therefore, when the power of the transmitter is switched off 8 records may be lost which corresponds to eight times the time span selected for a logging. In order to avoid this loss it is recommended to command a system reset by writing 1 to the Modbus register “Reset Request” (0x6008). Alternatively, the logging can be stopped by writing a 0 into Modbus register “RecordingRequest” (0x60D2).

In both cases a last record will be generated with bit 0 or 1 set in the “flags” field and then all remaining data will be written to the corresponding flash page. This generates up to 7 unused records in the last page.

When a recording starts it always starts on a new page boundary and its record id is dividable by 8 without remainder. This is one of the reasons why a previous logging sequence has to be searched backwards from the end of a logging sequence and requires a search span of at least 8 records.

Another reason for a widened search tolerance for the start and end points of logging sequences is the fact that NAND flash may have faulty memory portions which cannot be corrected by the integrated ECC memory protection. All records are protected by 16-bit CCITT CRCs which are used to ascertain that only correct data is read from the transmitter. Any CRC failure results in an error return of the corresponding read command. When the RHEComPro receives such an error return the respective data record simply is omitted in the generated CSV file. Such a gap may be detected by the user with the help of the time stamps or the written "record id" tags. Note, that gaps are normal at multiples of 512 in the record id because these records contain setup information. In this case, however, there will be no gap in the time stamps of the data records.

The number of faulty data portions which lead to a loss of a data records will be rather small. However, there are several consequences due to the possibility of faulty data portions:

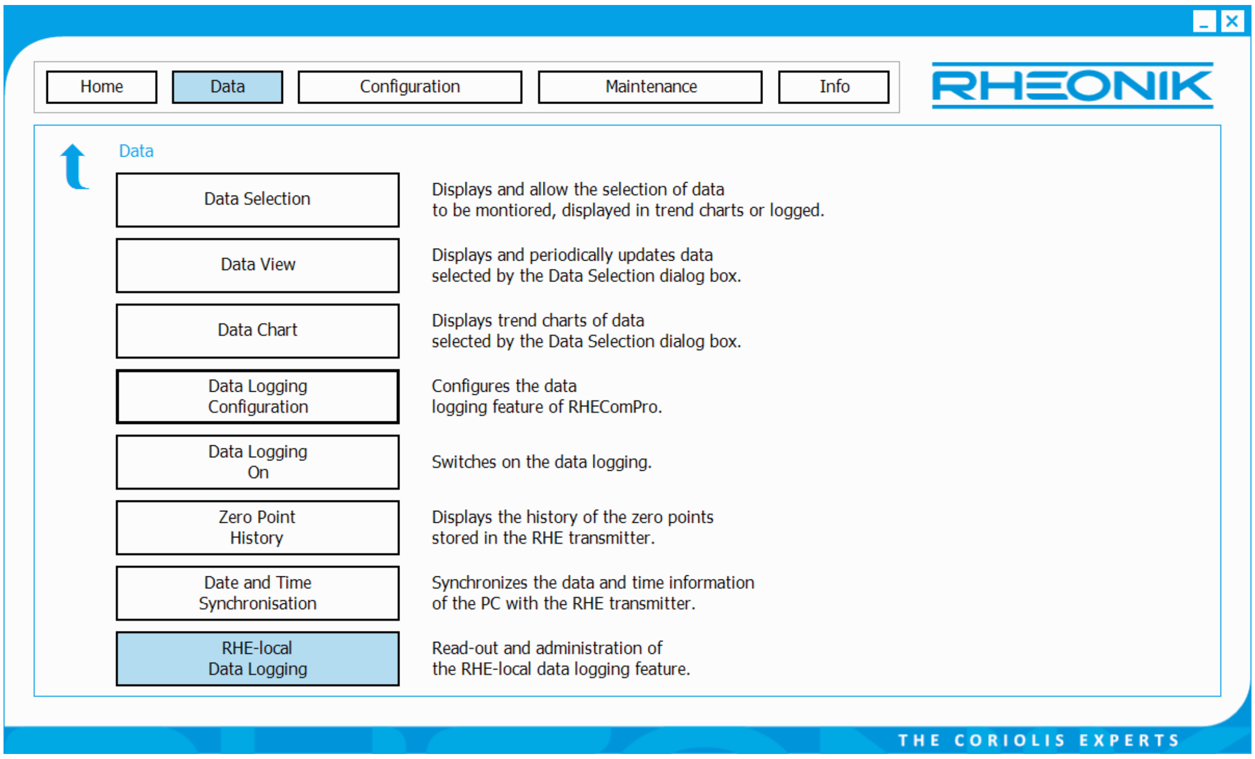
1. The search span for the start or end points of logging sequences should encompass at least a small multiple of 8 record ids. The current RHEComPro implementation uses a span of 26 ids.
2. The time interval for the logging should be chosen to half as long or less than it would be needed when all records can be recovered intact. It the potential loss of 8 records at the end of a recording due to a power failure matters the logging time span should even be set to one eighth or less of the needed time span.

3 User Interface to the Data Logging Feature

This section is intended for users who want to use the RHEComPro program to handle the Data Logging Feature and to generate a CSV file out of the data records.

3.1 Data Logging User Interface Box

When the RHEComPro program detects that is connected to an RHE4X series transmitter and was activated at least in the role "User" it offers a "RHE-local Data Logging" menu item in the selection of the "Data" menu as shown below.



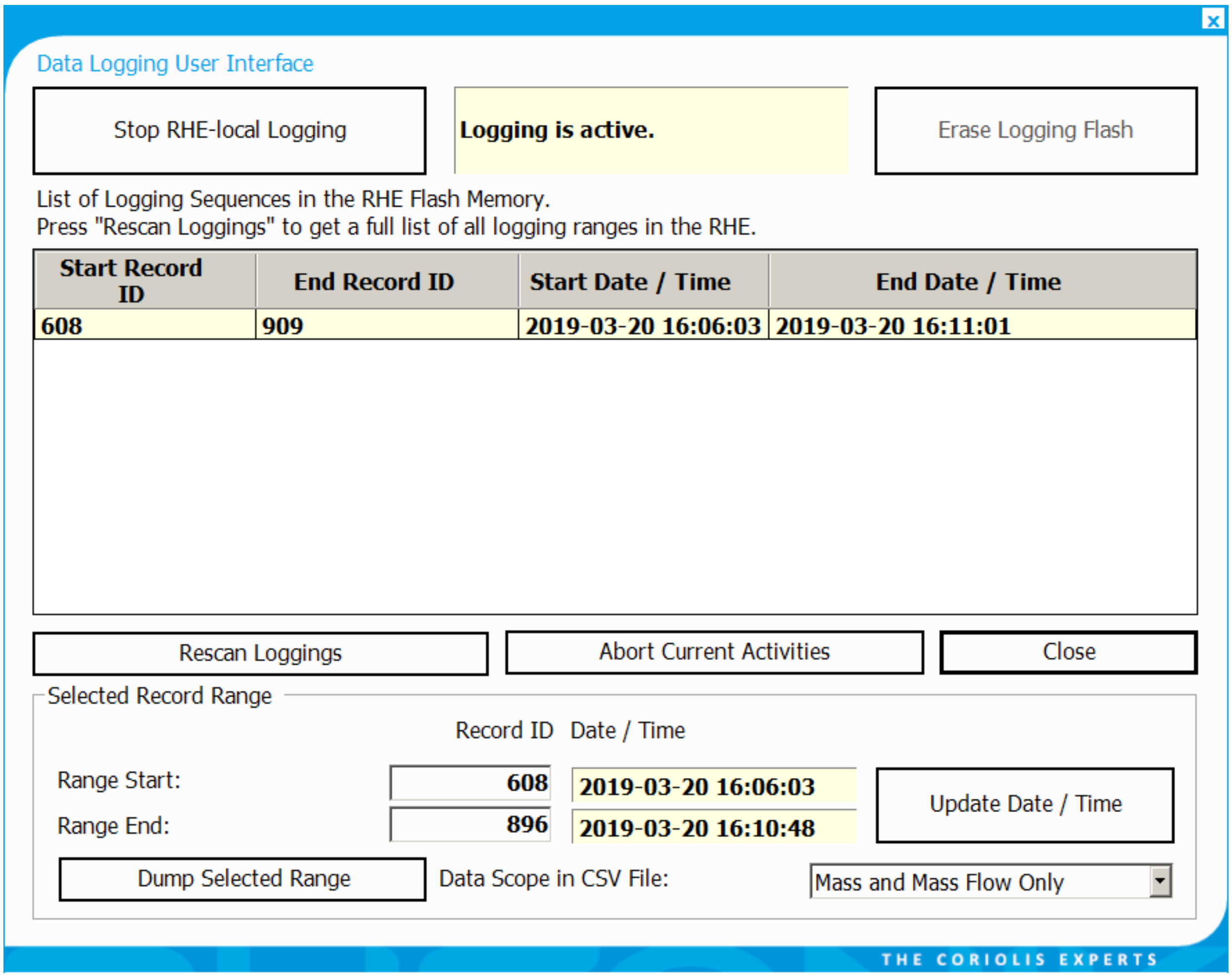
Data

Data Selection	Displays and allow the selection of data to be montiored, displayed in trend charts or logged.
Data View	Displays and periodically updates data selected by the Data Selection dialog box.
Data Chart	Displays trend charts of data selected by the Data Selection dialog box.
Data Logging Configuration	Configures the data logging feature of RHEComPro.
Data Logging On	Switches on the data logging.
Zero Point History	Displays the history of the zero points stored in the RHE transmitter.
Date and Time Synchronisation	Synchronizes the data and time information of the PC with the RHE transmitter.
RHE-local Data Logging	Read-out and administration of the RHE-local data logging feature.

Figure 1: “RHE-local Data Logging” Submenu Item in the “Data” Menu

When this menu item is selected the “Data Logging User Interface” dialog box appears. Aside from uploading recorded data from the RHE transmitter and the conversion of this data into a CSV file this dialog box offers several other functions concerning the “Data Logging Feature”.

When the dialog box appears only the properties of the current logging sequence is shown in the table in the center of the dialog box. This may be sufficient when only data of the current logging sequence has to be analyzed. The current state of the data logging is displayed in the text field at the top of the dialog box.



Logging is active.

List of Logging Sequences in the RHE Flash Memory.
Press "Rescan Loggings" to get a full list of all logging ranges in the RHE.

Start Record ID	End Record ID	Start Date / Time	End Date / Time
608	909	2019-03-20 16:06:03	2019-03-20 16:11:01

Selected Record Range

	Record ID	Date / Time
Range Start:	608	2019-03-20 16:06:03
Range End:	896	2019-03-20 16:10:48

Data Scope in CSV File: Mass and Mass Flow Only

THE CORIOLIS EXPERTS

Figure 2: "Data Logging User Interface" Dialog Box

Should the logging be stopped the displayed logging interval is empty because there is no data available. If previous logging sequences are present their properties will be displayed when the "Rescan Loggings" button is pressed. This causes the RHEComPro program to scan all available logging sequences and display them in the table.

Should there be more than one sequence in the table the row one of these sequences may be selected by a mouse click and the attributes of the sequence is transferred to the "Selected Record Range" group of dialog items. There the start and end Record Ids as well as the corresponding time stamps are displayed. Initially these properties indicate the entire span of a logging sequence from the start after a power cycle or a restart of the logging until the end of the logging which may be caused by a reset, a commanded system shutdown, or a stop the logging.

The number of records will amount to the difference of the end record id and the start record id. If this span contains too much records it is possible to narrow down the range of interest by modifying the start or end record ids. After a press on the "Update Date / Time" button the RHEComPro checks the plausibility of the new ids and tries to read the respective data records. Upon success updates the fields in the "Selected Record Range" group.

The selected range thus may be refined to cover a certain time span of interest. Before the data records are written to a CSV file the scope of the data to be written should be chosen in the "Data Scope" selection box. There are four scopes available:

Selection	Meaning
Mass and Mass Flow Only	This selection dumps the administrative data items like record ids and time stamps, status data like the SoftError bit field, all mass totalizers, and the mass flow.
Mass + Volume + Flows	Works like the previous selection, but adds the volume totalizers, the volume flow, and the density measurements to the data written.
Important Measurement Items	Works like the previous selection, but adds the temperature and pressure measurements to the data written.
Full Data Dump	Works like the previous selection, but adds diagnostic data and the state of the inputs and outputs to the data written. This setting is recommended when the logged data is to be used for a Rheonik Service request.

Table 4: Data Scope selection items

When the desired scope is selected the upload of the records and the generation of the CSV file can be started by pressing the “Dump Selected Range” button. A file selection box appears in which the target directory and target file may specified. After a successful file specification the upload of the data records starts.

Since this may be a lengthy operation it is possible to stop it by pressing the “Abort Current Activities” button

The logging inside the RHE can be started or stopped via the “Start Local Logging” available when the logging is stopped. This button which is converted to a “Stop Local Logging” functionality when the logging is active. Whenever a logging is stopped the current logging sequence is concluded and a subsequent re-start will create a new logging sequence. This start/stop state of the Data Logging Feature is stored in non-volatile RAM and will be applied when the system is restarted after a system reset or a power cycle. Thus, an active logging will be restarted when interrupted by a power failure, but this also will create a new logging sequence.

3.2 Description of the Generated CSV File

The amount of data columns in the CSV file depends on the “Data Scope” option in the “Data Logging User Interface” dialog box. Since the amount of data written to the file may be rather large it is recommend to restrict the data scope to the data items which are of interest. However, should the CSV file be intended to be part of a communication with the Rheonik service you should chose the “Full Data Dump” option in “Data Scope”. In the case the data record CSV file should be accompanied by a Setup CSV file which was generated by the “Save” function in the “Device Configuration (Factory)” dialog box.

The data items in the CSV file are separated by ‘;’ characters and the floating point numbers will feature a decimal point or a decimal comma depending on the “locale” settings of your windows operating system. Please check this with a text editor before importing the file into your Excel version. Whether the file format is accepted by your Excel installation also depends on the “locale” settings and options in the “Extended” tab of the Excel “File”/”Options” dialog box.

If these settings don’t match the format your Excel installation requires use a text editor first to replace all the decimal point characters and thereafter the data field separator characters. Both cannot be identical. Do this on a copy and keep the original file for further reference.

The first three lines of the CSV file constitute a header with following contents:

Line	**Contents**
1	Modbus Name of the data items as they are used in the Appendix of the RHE4X Desktop Reference Manual.
2	Hexadecimal Modbus register addresses of the data items as they are used in the Appendix of the RHE4X Desktop Reference Manual. These are convenient when searching for a description of the respective data item in this document.
3	Units for all items where applicable.

Table 5: Header Lines in the CSV File

The units used for the data items will in most cases be the user defined units currently installed in the RHE. These are not necessarily the units which were used when the data was recorded, but the unit conversion scheme makes sure that the data is correct when interpreted in accordance with the unit shown in the third line of the column of the respective data item.

The date/time stamp is written in column “D” as floating point number where the integer part corresponds to the number of days since 1900-01-01 fit for the Excel date/time handling. The corresponding time will be displayed when the format of these cells is set to one of the date or time formats. To obtain data and time information use one of the special time formats Excel offers. The correctness of these time stamps depends on the correctness of the time kept in the RHE-internal real-time clock (RTC).

Since this clock can be changed by the user (see Modbus “Generic” registers 0x60B0 to 0x60BD) or by network activities (see Modbus “Generic” register 0x60AE) there may be discontinuances in the time stamp column “D”. Whenever the time was changed the respective data record is marked by a set bit 3 (Mask 0x0008) in the “Flags” field written as hexadecimal number in column “C”. The “Flags” fields also records a number of events like the reset of the totalizers. See Table 3 for further information.

Since the date/time stamp column may contain discontinuances it is recommended to use the time data in column “A” as time reference for plots or data analysis. This time stamp type is a 32-bit unsigned counter which starts at 0 when the RHE transmits starts after a reset or a power cycle and is incremented every millisecond. Since this millisecond counter wraps every 50 days a logging may feature a rare discontinuance in column “A” where the number jumps from a value near 4294967295 to a value near 0. This can be remedied by adding the number 4294967296 to all time values after such discontinuance. The resulting values in column “A” then will provide a contiguous time reference based on milliseconds.

As described above discontinuances in the time line and thus in the data also may be caused by rare data corruptions in the NAND flash memory. Corrupt data records are omitted in the CSV file. This can be seen in the time reference in column “A” but also in the sequence of record ids in column “E” (except for setup information records at multiples of 512 in their ids). The amount of omitted records is the difference between the record id in a lower row and its adjacent upper row minus one.

When analyzing a system problem which is recognized by the transmitter and occurred during the time span covered by the logging the status information indicated in the “ErrorStatus” and “SoftError” fields in the columns “E” and “F” provides a valuable aid. At the moment of the occurrence of problem a change in the bit patterns of these fields can be expected. The interpretation of the hexadecimal numbers in these fields may be difficult to users who are not familiar with the conversion of hexadecimal numbers to their binary representations.

The hexadecimal number may be converted to binary numbers by Excel formulas in separate columns. For a German Excel version a conversion of an 8-digit hexadecimal number starting with “0x” and stored in the “E4” cell this would look like

```
=VERKETTEN(HEXINBIN(TEIL(E4;3;2);8);HEXINBIN(TEIL(E4;5;2);8);HEXINBIN(TEIL(E4;7;2);8);HEXINBIN(TEIL(E4;9;2);8))
```

This can be translated to English and other Excel versions, e.g. with the help of the Excel translator

https://de.excel-translator.de/translator/

The English translation results in

```
=CONCATENATE(HEX2BIN(MID(E4,3,2),8),HEX2BIN(MID(E4,5,2),8),HEX2BIN(MID(E4,7,2),8),HEX2BIN(MID(E4,9,2),8))
```

In the resulting string which consists of binary 0 and 1 digits a set bit is displayed as “1” and the bit position is determined by counting the character position from 0. E.g. in the 32-bit pattern

```
00000000000000000000000000001000
```

bit number 3 is set which would indicate a major Tube Temperature Sensor problem when present in the “SoftError” field. All relevant bits and their meanings are described in the Appendix referring to the “Generic” input registers within the RHE4X Desktop Reference Manual (See register addresses 0x401A to 0x4020.

Rheonik will attempt to develop further Excel formulas and aids which could be helpful in the analysis of the recorded data. Please check with the Rheonik Service regarding the current status of these aids. Please state your Windows “locale” or installation language in your request as Excel formulas do not have universal validity.

When the RHEComPro program managed to recover one of the setup records in the selected logging span it adds the data of the setup record in the first data row (4) after the data. The setup data will also feature all three header lines but will appear only once in order to save memory because this data will not change during on ongoing logging.

When the logging is stopped the “Erase Logging Flash” button is activated and can be used to erase the flash memory inside the RHE. As a consequence none of the previously existing logging sequences will be accessible and more. The erasure lasts a while and its termination may be observed by an update of the status text field on the top of the dialog box which is done every second. The logging may be restarted immediately after the erasure is commanded, but the actual start of the logging in the RHE transmitter is delayed until the erasure has terminated.

4 Detailed Description of the Software Interfaces

This section is intended for persons who want to write proprietary software in order to access the Data Logging Feature of the RHE transmitters. It also represents a documentation to Rheonik software developers who are not yet familiar with this features.

Since section 2 already provides an extensive description of the operation of the Data Logging Feature and the handling of the NAND flash memory this section concentrates on the details relevant to programmers.

4.1 Modbus Registers

See section 2 for a detailed description of the Modbus registers related to the Data Logging Feature.

4.2 RHE4X Data Logging Commands

The RHE4X Data Logging Commands are implemented as proprietary Rheonik Modbus Commands with the command code 0x72. The byte below the Modbus command code contains subcommands which constitute a whole set of commands reserved for the use in Rheonik transmitters.

Each data record is addressed via a “Record Id”, a 32-bit counter which is incremented when new data records are added to the recording. This Record Id number may be recalculated into a page- and block-indexed storage location of the NAND flash.

4.2.1 “Record Read” Subcommand

The “Record Read” command reads a portion of a data record in the NAND flash. This data record is identified by the “Record ID”. The data record is 256 Byte in size and due to the Modbus message size limits it needs at least two Record Read commands to read an entire record.

Byte	Meaning	Value
0	Modbus ID	1
1	Rheonik Command Code	0x72
2	Rheonik Subcommand “Record Read”	32
3 .. 6	Record ID	
7 .. 8	Byte Offset	0 to 255
9 .. 10	Byte Length	0 to 240
11	CRC low	
12	CRC high	

Following response is expected:

Byte	Meaning	Value
0	Modbus ID	1
1	Rheonik Command Code	0x72
2	Rheonik Subcommand “Record Read”	32
3 .. 6	Record Id	
7 .. 8	Byte Offset (M)	M
9 .. 10	Byte Length (N)	N
11 .. N + 10	Portion of Data Record in the range of M to M+N-1.	
N + 11	CRC low	
N + 12	CRC high	

The Data Record is filled with binary data in little-endian format whilst the remaining fields of the Modbus command are represented in big-endian (network) byte order. The structure of the record entire 256 Byte record is detailed in Appendix A and Appendix B.

The issuer of the “Record Read” command must be aware that not all Record Ids exist. The “Generic” Modbus input registers RecordingMinId (0x4034) and RecordingMaxId (0x4036) give an indication of the currently valid range of “Record Ids”. However, not all of these records may exists, due to following reasons:

- There is a small gap of maximum 8 records between a stop of loggings and a restart of the next logging interval.
- Records may reside in NAND flash areas which are subject to data corruption.

In order to distinguish the reasons for an error return in the “Record Read” response following Modbus error codes are used:

Code	Name	Meaning
02	ILLEGAL DATA ADDRESS	The offset/length parameter point outside the record or would result in a Modbus response which is too large.
03	ILLEGAL DATA VALUE	The Record Id does not exist. Do not retry this command.
04	SERVER DEVICE FAILURE	An unrecoverable read error (ECC or CRC) occurred during an access to the record data. Do not retry this command.
06	SERVER DEVICE BUSY	NAND Flash currently busy with a longer operation. Retry this command.

4.2.2 “Logging Erase” Subcommand

The “Logging Erase” subcommand is used to erase the entire NAND flash inside the RHE transmitter that is used for logging purposes. This command has following structure:

Byte	Meaning	Value
0	Modbus ID	1
1	Rheonik Command Code	0x72
2	Rheonik Subcommand “Logging Erase”	33
3	CRC low	
4	CRC high	

Following response is expected:

Byte	Meaning	Value
0	Modbus ID	1
1	Rheonik Command Code	0x72
2	Rheonik Subcommand “Logging Erase”	33
3	Status Return: 0: Erase started. 1: Erase already running. 2: Flash busy, retry command. 0xFF: Erase command rejected. Check logging status (0x603E).	
4	CRC low	
5	CRC high	

The erase command can only be executed successfully when the logging is stopped.

Appendix

Appendix A C Structures used for the Logging Feature

This section contains the structures used for the Data Logging Feature implemented in C. The structure *recording_layout_setup* is present when the bit *RECORDING_FLAGS_SETUP_RECORD* is set in the *flags* field. This type of record contains relevant setup parameters and is recorded at the very start of the logging and at the start of each NAND block, usually every 512 records. Note, that the start of the loggings indicated by *reset_recording_id* may already be overwritten by new data records.

When the bit the bit *RECORDING_FLAGS_SETUP_RECORD* is not present in the *flags* field the data records contains the structure *recording_layout* which contains all the status and the measurement information present in the RHE transmitter.

The data in records is stored in little-endian fashion and thus need not be swapped when transferred to a standard PC. The alignment is fit for compilers which store 4-Byte and 8-Byte types on 4-byte address boundaries. When a compiler is used which stores 8-Byte types on 8-Byte address boundaries the *recording_layout* structure must be defined as “packed”.

The data type float including the derived types describes a 32-bit IEEE floating point number and double and its derived types represent a 64-bit IEEE floating point number. The remaining type definitions should be self-explanatory.

The hexadecimal numbers in the comments behind record fields define the Modbus register from which the respective data item usually can be read. The description of the respective data item can be found in the Appendix of the “RHE4X Desktop Reference Manual”, Document Number 8.2.1.14, when using this number as search key.

```
typedef signed char int8_t;
typedef signed short int16_t;
typedef signed long int int32_t;
typedef unsigned char uint8_t;
typedef unsigned short uint16_t;
typedef unsigned long int uint32_t;
typedef float ieee_float;
typedef double ieee_double;
typedef ieee_float float32_t;
typedef ieee_double float64_t;

#define RECORDING_RECORD_SIZE               (256U)  /* in Bytes */

struct recording_layout
{
    /* Offset 0x0000: */
    uint16_t crc;

#define RECORDING_FLAGS_START_AFTER_RESET   ((uint16_t) 0x0001U)
#define RECORDING_FLAGS_STOPPED             ((uint16_t) 0x0002U)
#define RECORDING_FLAGS_RESTARTED           ((uint16_t) 0x0004U)
#define RECORDING_FLAGS_TIME_CHANGED        ((uint16_t) 0x0008U)
#define RECORDING_FLAGS_TOTALIZER_RESET     ((uint16_t) 0x0010U)
#define RECORDING_FLAGS_TOTALIZER_STOPPED   ((uint16_t) 0x0020U)
#define RECORDING_FLAGS_SHUTDOWN_COMMANDED  ((uint16_t) 0x0040U)
#define RECORDING_FLAGS_ZEROING_COMMANDED   ((uint16_t) 0x0080U)
#define RECORDING_FLAGS_SETUP_RECORD        ((uint16_t) 0x8000U)
    uint16_t flags;

    /* Offset 0x0004: */
    uint32_t record_id;             /* Wrap-around counter. */
    uint32_t reset_record_id;       /* Start of recording @ last reset. */

    /* Offset 0x000C: */
    uint32_t time_stamp;            /* Local (RTC) Time since 01.01.1980 0:00 */
    uint32_t time_since_reset;      /* Seconds since reset */

    /* Offset 0x0014: */            /* System status: */
    uint32_t ErrorStatus;           /* 401A */
    uint32_t SoftError;             /* 401C */
    uint32_t Warnings;              /* 401E */
    uint32_t InfoStatus;            /* 4020 */

    /* Offset 0x0024: */            /* Totalizers: */
    ieee_double TotInvenMassNet;    /* 4B04 */
    ieee_double TotInvenVolNet;     /* 4B06 */
    ieee_double TotalMassFwd;       /* 4B00 */
    ieee_double TotalVolFwd;        /* 4B02 */
    ieee_double TotalMassRev;       /* 4B08 */
    ieee_double TotalVolRev;        /* 4B02 */
    ieee_double SecTotNetMass;      /* 4B2C */
    ieee_double SecTotNetVolume;    /* 4B30 */

    /* Offset 0x0064: */            /* Mass Flow: */
    ieee_float MassFlowRateModbus;  /* 4908 */
    ieee_float VolFlowRateModbus;   /* 4A06 */

    ieee_float AdcTubeMeanTemp;     /* 4500 */
    ieee_float AdcTorBarMeanTemp;   /* 4502 */

    /* Offset 0x0074: */
    ieee_float OnBrdTemp;           /* 4504 */
    ieee_float DenComp;             /* 4806 */
    ieee_float StdDensity;          /* 480A */
    ieee_float CutMainMass;         /* 480E */

    /* Offset 0x0084: */
    ieee_float VolPercentMainSubstance; /* 480C */
    ieee_float VolFlwNorDensCurr;   /* 6838 */

    ieee_float PrsMean;             /* 4606 */
    ieee_float SensorFrequency;     /* 4206 */
    /* Offset 0x0094: */
    int16_t AnOutputStage;          /* 4400 */
    uint16_t AnInputLeftCoil;       /* 4404 */
    uint16_t AnInputRightCoil;      /* 4406 */
    uint16_t DriveGain;             /* 440E in percent*/
    ieee_float DriveCurrentmA;      /* 440C */
    ieee_float AssuranceFactor;     /* 4026 */

    /* Offset 0x00A4: */
    uint8_t DigiOutChAlmState [4];  /* 4D04, 4D06, 4E04, 4E06, */
    uint8_t DIMirror [2];           /* 4F02, 4F04 */
    uint8_t reserved1 [2];
    ieee_float CurrOut [2];         /* 4C00, 4C02 */

    ieee_float ZeroPointPhase;      /* 671A */

    /* Offset 0x00B8: */
    ieee_float MassFlowRateNoCutOff; /* 490A */

    /* Offset 0x00BC: */
    uint32_t reserved2 [(RECORDING_RECORD_SIZE - 0xBCU) / sizeof (uint32_t)];
};


/* Must be identical to recording_layout up to
 * offset 0x14.
 */
struct recording_layout_setup
{
    /* Offset 0x0000: */
    uint16_t crc;
    uint16_t flags;

    /* Offset 0x0004: */
    uint32_t record_id;                 /* Wrap-around counter. */
    uint32_t reset_record_id;           /* Start of recording @ last reset. */

    /* Offset 0x000C: */
    uint32_t time_stamp;                /* Local (RTC) Time since 01.01.1980 0:00 */
    uint32_t time_since_reset;          /* Seconds since reset */

    /* Offset 0x0014: */                /* Generic: */
    uint32_t   SensorType;              /* 601A */

    uint8_t    AssurancePresent;        /* 6090 */
    uint8_t    VolDensPresent;          /* 6084 */
    uint8_t    RS485Present;            /* 60XX */
    uint8_t    CurrOutPresent;          /* 6086 */

    uint16_t   DigOutPresent;           /* 6088 */
    uint8_t    APIDnsPresent;           /* 6092 */
    uint8_t    CurrInputPresent;        /* 608A */

    uint8_t    HARTPresent;             /* 608C */
    uint8_t    RHEType;                 /* 608E */

    uint16_t   FreqFilNoSamples;        /* 6208 */

    /* Offset 0x0024: */                /* Ampl Diagnostic: */
    ieee_float OutputCtlTargetPickup;   /* 640A */
    ieee_float OutputCtlIntegralTarget; /* 640C */
    ieee_float OutputCtlPropFactor;     /* 640E */
    ieee_float OutputCtlIntFactor;      /* 6410 */

    ieee_float OutputCtlDiffFactor;     /* 6412 */
    ieee_float OutputCtlPhaseOffset;    /* 6414 */

    uint8_t PhsFlwDirConfig;            /* 6308 */
    uint8_t PhsDSPMethod;               /* 636C */
    uint16_t PhsFilNoSamples;           /* 630A */
    ieee_float FlowFilterDisplayTau;    /* 6366 */
    ieee_float FlowFilterFreqTau;       /* 6368 */
    ieee_float FlowFilterModbusTau;     /* 636A */

    /* Offset 0x004C: */                /* Mass Flow: */
    ieee_float MsFlwTubeRefTemp;        /* 690A */
    ieee_float MsFlwTorBarRefTemp;      /* 690C */
    ieee_float s10;                     /* 6910 */
    ieee_float s01;                     /* 6912 */
    ieee_float MassFlowKFactor;         /* 6922 */
    ieee_float MassFlowCutOffLimit;     /* 6924 */
    ieee_float TempCorSTD;              /* 693A */

    /* Offset 0x0068: */                /* Density: */
    uint8_t dnsConfig;                  /* 6800 */
    uint8_t DenCalcMode;                /* 683A */
    uint16_t den_reserved;
    ieee_float DnsTubeRefTemp;          /* 680E */
    ieee_float DnsTorBarRefTemp;        /* 6810 */
    ieee_float u10;                         /* 6814 */

    ieee_float u01;                     /* 6816 */
    ieee_float dnsLowDensityCalPoint;   /* 6826 */
    ieee_float dnsLowDensityFrequency;  /* 6828 */
    ieee_float dnsHighDensityCalPoint;  /* 682A */

    ieee_float dnsHighDensityFrequency; /* 682C */
    ieee_float VolFlwNorDens;           /* 6832 */
    ieee_float dnsRefTmpNorDns;         /* 6834 */
    ieee_float dnsTmpCoeff;             /* 6836 */

    ieee_float DenMainSubstance;        /* 683C */
    ieee_float DenAddSubstance;         /* 683E */

    /* Offset 0x00A0: */                /* Temperature: */
    uint16_t TempConfig;                /* 6500 */
    uint16_t AdcTubeFilNoSamples;       /* 6516 */
    uint16_t AdcTorBarFilNoSamples;     /* 6518 */
    uint16_t temp_reserved;
    ieee_float AdcTubeOffset;           /* 6512 */
    ieee_float AdcTorBarOffset;         /* 6514 */

    ieee_float AdcTubeCalOffset;        /* 651A */
    ieee_float AdcTubeCalGain;          /* 651C */
    ieee_float AdcTorBarCalOffset;      /* 651E */
    ieee_float AdcTorBarCalGain;        /* 6520 */

    /* Offset 0x00C0: */                /* Pressure: */
    uint16_t PressureCalcConfig;        /* 6610  */
    uint16_t  AdcFilNoSamples;          /* 6608  */
    ieee_float PrsValMin;               /* 6604  */
    ieee_float PrsValMax;               /* 6606  */
    ieee_float PrsOffset;               /* 660E  */

    ieee_float PrsExternalInitial;      /* 6612  */
    uint32_t AdcCalOffset;              /* 6618  */
    uint32_t AdcCalGain;                /* 661A  */
    ieee_float DnsValMin;               /* 6622  */
    ieee_float DnsValMax;               /* 6624  */

    /* Offset 0x00E4: */                /* Generic: */
    float32_t  variancePhase;           /* 6724 */
    float32_t  variancePeriod;          /* 6726 */
    uint32_t   ZeroingTimeStamp;        /* TODO @@**/

    uint16_t   ZeroingNumberOfSamples;  /* 6728 */

    /* Offset 0x00F2: */                /* HMI: */
    uint16_t BatchMode;                 /* 6F0E */

    /* Offset 0x00F4: */                /* HMI: */
    uint16_t DIProperty [2];            /* 6F0A */

    /* Offset 0x00F8: */
    uint32_t reserved3 [(RECORDING_RECORD_SIZE - 0xF8U) / sizeof (uint32_t)];
};
```

Appendix B C# Structure used for the Data Logging Feature

This section contains the structures used for the Data Logging Feature implemented in C#. The structure contains two layouts defined with the help of the C# *[FieldOffset(x)]* feature, see also Appendix A. The second structure layout starting with *SensorType* is present when the bit *RECORDING_FLAGS_SETUP_RECORD* is set in the flags field. This type of record contains relevant setup parameters and is recorded at the very start of the recording and at the start of each NAND block, usually every 512 records. Note, that the start of the records indicated by *reset_record_id* may already be overwritten by new data records.

When the bit the bit *RECORDING_FLAGS_SETUP_RECORD* is not present in the flags field the data recordings contains the first structure layout starting with the ErrorStatus field. This structure layout contains all the status and the measurement information present in the RHE transmitter.

The data in records is stored in little-endian fashion and thus need not be swapped when transferred to a standard PC. The alignment is defined by the *[FieldOffset(x)]* declarations. Only standard C# data types are used. The array *byte_array* is intended to facilitate the transfer of raw data buffers to the *recording_layout* structure, see the comment above the respective declaration.

The hexadecimal numbers in the comments behind record fields define the Modbus register from which the respective data item usually can be read. The description of the respective data item can be found in the Appendix of RHE4X Desktop Reference Manual, Document Number 8.2.1.14, when using this number as search key.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Runtime.InteropServices;

…

    /* Masks for the "flags" data item in recording_layout. */
    public const UInt16 RECORDING_FLAGS_START_AFTER_RESET   = 0x0001;
    public const UInt16 RECORDING_FLAGS_STOPPED             = 0x0002;
    public const UInt16 RECORDING_FLAGS_RESTARTED           = 0x0004;
    public const UInt16 RECORDING_FLAGS_TIME_CHANGED        = 0x0008;
    public const UInt16 RECORDING_FLAGS_TOTALIZER_RESET     = 0x0010;
    public const UInt16 RECORDING_FLAGS_TOTALIZER_STOPPED   = 0x0020;
    public const UInt16 RECORDING_FLAGS_SHUTDOWN_COMMANDED  = 0x0040;
    public const UInt16 RECORDING_FLAGS_ZEROING_COMMANDED   = 0x0080;
    public const UInt16 RECORDING_FLAGS_SETUP_RECORD        = 0x8000;

    [StructLayout(LayoutKind.Explicit, Size = 256)]
    public struct recording_layout
    {
        /*
         * byte_array is intended to be used in an "unsafe"
```

```
 * copy loop to fill the record, e.g.:
 *      unsafe
 *      {
 *          fixed (byte * dest = &record.byte_array)
 *          for (int i = 0; i < 128; i++)
 *          {
 *              dest [i] = buf [i];
 *           }
 *       }
 */
[FieldOffset(0)]
public byte byte_array;

[FieldOffset(0)]
public UInt16 crc;

[FieldOffset(2)]
public UInt16 flags;

[FieldOffset(4)]
public UInt32 record_id;

[FieldOffset(8)]
public UInt32 reset_record_id;

[FieldOffset(12)]
public UInt32 time_stamp;

[FieldOffset(16)]
public UInt32 time_since_reset;

[FieldOffset(20)]
public UInt32 ErrorStatus;          /* 401A */

[FieldOffset(24)]
public UInt32 SoftError;            /* 401C */

[FieldOffset(28)]
public UInt32 Warnings;             /* 401E */

[FieldOffset(32)]
public UInt32 InfoStatus;           /* 4020 */

[FieldOffset(36)]
public double TotInvenMassNet;      /* 4B04 */
[FieldOffset(44)]
public double TotInvenVolNet;       /* 4B06 */
[FieldOffset(52)]
public double TotalMassFwd;         /* 4B00 */
[FieldOffset(60)]
public double TotalVolFwd;          /* 4B02 */
[FieldOffset(68)]
public double TotalMassRev;         /* 4B08 */
[FieldOffset(76)]
public double TotalVolRev;          /* 4B02 */
[FieldOffset(84)]
public double SecTotNetMass;        /* 4B2C */
[FieldOffset(92)]
public double SecTotNetVolume;      /* 4B30 */

[FieldOffset(100)]
public float MassFlowRateModbus;    /* 4908 */

[FieldOffset(104)]
public float VolFlowRateModbus;     /* 4A06 */

[FieldOffset(108)]
public float AdcTubeMeanTemp;       /* 4500 */
[FieldOffset(112)]
```

```
public float AdcTorBarMeanTemp;     /* 4502 */

[FieldOffset(116)]
public float OnBrdTemp;             /* 4504 */
[FieldOffset(120)]
public float DenComp;               /* 4806 */
[FieldOffset(124)]
public float StdDensity;            /* 480A */
[FieldOffset(128)]
public float CutMainMass;           /* 480E */

[FieldOffset(132)]
public float VolPercentMainSubstance; /* 480C */
[FieldOffset(136)]
public float VolFlwNorDensCurr;     /* 6838 */

[FieldOffset(140)]
public float PrsMean;               /* 4606 */
[FieldOffset(144)]
public float SensorFrequency;       /* 4206 */

[FieldOffset(148)]
public Int16 AnOutputStage;         /* 4400 */
[FieldOffset(150)]
public UInt16 AnInputLeftCoil;      /* 4404 */
[FieldOffset(152)]
public UInt16 AnInputRightCoil;     /* 4406 */
[FieldOffset(154)]
public UInt16 DriveGain;            /* 440E in percent*/
[FieldOffset(156)]
public float DriveCurrentmA;        /* 440C */
[FieldOffset(160)]
public float AssuranceFactor;       /* 4026 */

[FieldOffset(164)]
public byte DigiOutChAlmState1;     /* 4D04 */
[FieldOffset(165)]
public byte DigiOutChAlmState2;     /* 4D06 */
[FieldOffset(166)]
public byte DigiOutChAlmState3;     /* 4E04 */
[FieldOffset(167)]
public byte DigiOutChAlmState4;     /* 4E06 */
[FieldOffset(168)]
public byte DIMirror1;              /* 4F02 */
[FieldOffset(169)]
public byte DIMirror2;              /* 4F04 */
[FieldOffset(170)]
public byte reserved1_a;
[FieldOffset(171)]
public byte reserved1_b;
[FieldOffset(172)]
public float CurrOut1;              /* 4C00 */
[FieldOffset(176)]
public float CurrOut2;              /* 4C02 */

[FieldOffset(180)]
public float ZeroPointPhase;        /* 671A */

[FieldOffset(184)]
public float MassFlowRateNoCutOff;  /* 490A */

/* setup record - starts @ offset 20 */
[FieldOffset(20)]
public UInt32 SensorType;           /* 601A */

[FieldOffset(24)]
public byte AssurancePresent;       /* 6090 */
[FieldOffset(25)]
public byte VolDensPresent;         /* 6084 */
```

```
[FieldOffset(26)]
public byte RS485Present;           /* 6094 */
[FieldOffset(27)]
public byte CurrOutPresent;         /* 6086 */

[FieldOffset(28)]
public UInt16 DigOutPresent;        /* 6088 */
[FieldOffset(30)]
public byte APIDnsPresent;          /* 6092 */
[FieldOffset(31)]
public byte CurrInputPresent;       /* 608A */
[FieldOffset(32)]
public byte HARTPresent;            /* 608C */
[FieldOffset(33)]
public byte RHEType;                /* 608E */

[FieldOffset(34)]
public UInt16 FreqFilNoSamples;     /* 6208 */

[FieldOffset(36)]
public float OutputCtlTargetPickup; /* 640A */
[FieldOffset(40)]
public float OutputCtlIntegralTarget; /* 640C */
[FieldOffset(44)]
public float OutputCtlPropFactor;   /* 640E */
[FieldOffset(48)]
public float OutputCtlIntFactor;    /* 6410 */

[FieldOffset(52)]
public float OutputCtlDiffFactor;   /* 6412 */
[FieldOffset(56)]
public float OutputCtlPhaseOffset;  /* 6414 */

[FieldOffset(60)]
public byte PhsFlwDirConfig;        /* 6308 */
[FieldOffset(61)]
public byte PhsDSPMethod;           /* 636C */
[FieldOffset(62)]
public UInt16 PhsFilNoSamples;      /* 630A */
[FieldOffset(64)]
public float FlowFilterDisplayTau;  /* 6366 */
[FieldOffset(68)]
public float FlowFilterFreqTau;     /* 6368 */
[FieldOffset(72)]
public float FlowFilterModbusTau;   /* 636A */

[FieldOffset(76)]
public float MsFlwTubeRefTemp;      /* 690A */
[FieldOffset(80)]
public float MsFlwTorBarRefTemp;    /* 690C */
[FieldOffset(84)]
public float s10;                   /* 6910 */
[FieldOffset(88)]
public float s01;                   /* 6912 */
[FieldOffset(92)]
public float MassFlowKFactor;       /* 6922 */
[FieldOffset(96)]
public float MassFlowCutOffLimit;   /* 6924 */
[FieldOffset(100)]
public float TempCorSTD;            /* 693A */

[FieldOffset(104)]
public byte dnsConfig;              /* 6800 */
[FieldOffset(105)]
public byte DenCalcMode;            /* 683A */
[FieldOffset(106)]
public UInt16 den_reserved;
[FieldOffset(108)]
public float DnsTubeRefTemp;        /* 680E */
```

```
[FieldOffset(112)]
public float DnsTorBarRefTemp;      /* 6810 */
[FieldOffset(116)]
public float u10;                   /* 6814 */
[FieldOffset(120)]
public float u01;                   /* 6816 */
[FieldOffset(124)]
public float dnsLowDensityCalPoint; /* 6826 */
[FieldOffset(128)]
public float dnsLowDensityFrequency;/* 6828 */
[FieldOffset(132)]
public float dnsHighDensityCalPoint;/* 682A */
[FieldOffset(136)]
public float dnsHighDensityFrequency; /* 682C */
[FieldOffset(140)]
public float VolFlwNorDens;         /* 6832 */
[FieldOffset(144)]
public float dnsRefTmpNorDns;       /* 6834 */
[FieldOffset(148)]
public float dnsTmpCoeff;           /* 6836 */
[FieldOffset(152)]
public float DenMainSubstance;      /* 683C */
[FieldOffset(156)]
public float DenAddSubstance;       /* 683E */

[FieldOffset(160)]
public UInt16 TempConfig;           /* 6500 */
[FieldOffset(162)]
public UInt16 AdcTubeFilNoSamples;  /* 6516 */
[FieldOffset(164)]
public UInt16 AdcTorBarFilNoSamples;/* 6518 */
[FieldOffset(166)]
public UInt16 temp_reserved;
[FieldOffset(168)]
public float AdcTubeOffset;         /* 6512 */
[FieldOffset(172)]
public float AdcTorBarOffset;       /* 6514 */
[FieldOffset(176)]
public float AdcTubeCalOffset;      /* 651A */
[FieldOffset(180)]
public float AdcTubeCalGain;        /* 651C */
[FieldOffset(184)]
public float AdcTorBarCalOffset;    /* 651E */
[FieldOffset(188)]
public float AdcTorBarCalGain;      /* 6520 */

[FieldOffset(192)]
public UInt16 PressureCalcConfig;   /* 6610  */
[FieldOffset(194)]
public UInt16 AdcFilNoSamples;      /* 6608  */
[FieldOffset(196)]
public float PrsValMin;             /* 6604  */
[FieldOffset(200)]
public float PrsValMax;             /* 6606  */
[FieldOffset(204)]
public float PrsOffset;             /* 660E  */

[FieldOffset(208)]
public float PrsExternalInitial;    /* 6612  */
[FieldOffset(212)]
public UInt32 AdcCalOffset;         /* 6618  */
[FieldOffset(216)]
public UInt32 AdcCalGain;           /* 661A  */
[FieldOffset(220)]
public float DnsValMin;             /* 6622  */
[FieldOffset(224)]
public float DnsValMax;             /* 6624  */
 [FieldOffset(228)]
public float variancePhase;         /* 6724 */
```

```
    [FieldOffset(232)]
    public float variancePeriod;        /* 6726 */
    [FieldOffset(236)]
    public UInt32 ZeroingTimeStamp;     /* */
    [FieldOffset(240)]
    public UInt16 ZeroingNumberOfSamples;  /* 6728 */

    [FieldOffset(242)]
    public UInt16 BatchMode;            /* 6F0E */

    [FieldOffset(244)]
    public UInt16 DIProperty1;          /* 6F0A */
    [FieldOffset(246)]
    public UInt16 DIProperty2;          /* 6F0C */
}
```

GET FLOW MEASURED

About Rheonik

Rheonik has but one single purpose: to design and manufacture the very best Coriolis meters available.

Our research and engineering resources are dedicated to finding new and better ways to provide cost effective accurate mass flow solutions that provide value to our customers. Our manufacturing group care for each and every meter we produce from raw materials all the way to shipping, and our service and support group are available to help you specify, integrate, start-up and maintain every Rheonik meter you have in service. Whether you own just one meter or have hundreds, you will never be just another customer to us. You are our valued business partner.

Need a specific configuration for your plant? Don't compromise with a "standard" product from elsewhere that will add extra cost to your installation. If we can't configure it from our extensive and versatile product range, our exclusive **AnyPipeFit Commitment** can have your flow sensor customized with any size/type of process connection and face to face dimension you need.

No matter what control system you use as the backbone in your enterprise, with our **AnyInterface Commitment**, you can be sure that connection and communication will not be a problem. Alongside a wide variety of discrete analog and digital signal connections, we can also provide just about any network/bus interface available (for example: HART, ProfibusDP, ProfiNet, EtherCAT, PowerLink, EtherNet/IP, CAN, ….) with our RHE 40 Series family of transmitters. Rheonik RHE 40 Series transmitters can connect to your system – no headache and no conversion needed.

Rheonik Messtechnik GmbH
Rudolf-Diesel-Straße 5
D-85235 Odelzhausen
Germany

Tel. + 49 (0)8134 9341-0
info@rheonik.com