

Novar1xxx-line PF-controller remote link communication description

Programmer handbook

01/2015

Power factor controllers of types Novar-1106/1114/1206/1214/1312 can be equipped with RS-232 or RS-485 remote link. Controller operation can be monitored and controlled over the remote link from supervising system (master, usually a PC).

This handbook describes the communication from point of view of application programmer. Basic knowledge of controller parameters and C-language syntax is supposed.

1.1 Data structures

The data interchanged between controller and master are organized to following structures :

- „Status“ ... contains actual controller state information (type, serial number, state of outputs, alarms, errors etc.); is read-only
- „EEStatus“ ... contains another state information such like maximum registered values, number of switching operations and switch-on time of all outputs etc.; is read-only
- „NovarStatus“ ... contains basic information of controller state and actual values of quantities measured; especially for on-line visualisation purposes; is read-only
- „Config“ ... contains actual state of controller parameters; can be both read and written
- „NovarSetMap“ ... virtual structure, that serves for selected controller functions' starting; is write-only

The master can get information of controller state by reading a structure. By writing to appropriate structure it can change any of parameters, start some of controller function etc.

Description of the structures follows in separate chapter below.

1.2 Communication protocols

The communication between a master and the controller (slave) runs over asynchronous serial link (COM) with RS-232 or RS-485 interface. If the RS-485 interface is used several controllers can be connected to a single link. At the master side a RS-232/RS-485 converter with automatic transmission direction switch can be used, or the converter direction need to be controlled by the master program.

The controllers are shipped with proprietary “KMB” protocol preset as default. Optionally the Modbus-RTU protocol can be set.

The communication rate can be set according controller technical parameters.

1.2.1 KMB communication protocol

The communication channel is set to 8 bits, no parity, one stop-bit.

Communication type is “Master-Slave”, i.e. after receiving message-command from master (PC) the controller processes it and transmits message-answer back to the master.

The message format is as follows :

1. Device address (1 byte)
2. Message length (in bytes) without final message checksum. (1 byte). It has the value of (3 + message body length).
3. Message type (1 byte).
4. Message body - its length differs each from other according the message type. Some messages can have no body (i.e. these messages’ body length is zero).
5. Message checksum (1 byte) - sum of all preceding bytes modulo 256 (1 byte).

When receiving correct command, the slave (controller) processes it and transmits answer. If it can process the command successfully, the message-type value in the answer is cleared to zero. Otherwise the message-type value contains error-specification.

The slave must answer during 600 ms after receiving a command from the master, available gap of up to 4 bytes transmit time between bytes during transmitting can occur.

1.2.1.1 Message types

Following message types can be used for reading/writing data structures :

message no. (hex)	message type
0x14	Device Status Read - Status, EEStatus
0x16	Device Setting Read – Config
0x17	Device Setting Write - Config
0x30	Device Status Read – NovarStatus
0x31	Device Function Start – NovarSetMap Write

1.2.1.1.1 Device Status Read (Status, EEStatus) – 0x14

Message no. 0x14. The slave returns in answer the Status (34 bytes long) and the EEStatus (110 bytes) structures, i.e. 144 bytes.

Example :

Master must send following sequence of bytes (address value 1 is supposed) :

| 0x01| 0x03 | 0x14 | checksum = 0x18 |

The command has no body.

Answer :

| 0x01| 0x93| 0x00 | message body = Status+EEStatus (144 bytes) | checksum |

(2nd byte, i.e. message length without checksum, is 144 + 3 = 147 = 0x93).

1.2.1.1.2 Device Setting Read (Config) – 0x16

Message no. 0x16. The slave returns the Config structure (100 bytes).

Command :

| 0x01| 0x03 | 0x16 | checksum = 0x1A |

Answer :

| 0x01| 0x67| 0x00 | message body = Config (80 bytes) | checksum |

1.2.1.1.3 Device Setting Write (Config) – 0x17

Message no. 0x17. The master can write the Config structure (80 bytes) to slave with this command.

Command :

| 0x01| 0x67 | 0x17 | message body = Config (80 bytes) | checksum |

Answer :

| 0x01| 0x03| 0x00 | checksum = 0x04 |

Comment : DeviceAddr and RemoteBDRate variables cannot be set over communication link, the values written to the variables are arbitrary.

1.2.1.1.4 Device Status Read (NovarStatus) – 0x30

Message no. 0x30. The slave returns state information in the NovarStatus structure (60 bytes), necessary especially for on-line monitoring and visualisation purposes.

Command :

| 0x01| 0x03 | 0x30 | checksum = 0x34 |

Answer :

| 0x01| 0x3F| 0x00 | message body = NovarStatus (60 bytes) | checksum |

1.2.1.1.5 Device Function Start (with NovarSetMap) – 0x31

Message no. 0x31. Some of controller functions can be started by writing to the NovarSetMap virtual structure (6 bytes). The functions are selected by setting appropriate values to 1.

Command :

| 0x01| 0x09 | 0x31 | message body = NovarSetMap (6 bytes) | checksum |

Answer :

| 0x01| 0x03| 0x00 | checksum = 0x04 |

1.2.2 Modbus-RTU protocol

Excluding the 1312 and the 1414 models, the standard Modbus-RTU communication protocol can be used optionally for connection with master. Except slave address a and communication rate, parity bit function can be set (even / odd / none parity).

The slave must answer during 600 ms after receiving a command from the master, available gap of up to 1.5 bytes transmit time between bytes during transmitting can occur.

Broadcast-mode is not supported.

Implemented Modbus functions are listed in following table :

function no.	function	application
03	Read Holding Registers	Config read – registers 40101-40140(addressed 100 – 139)
04	Read Input Registers	Status+EEStatus read – registers 30101-30172 (addressed 100 – 171) NovarStatus read – registers 30201-30230 (addressed 200 – 229)
06	Preset Single Register	Config write – registers 40101-40150 (addressed 100 – 149) NovarSetMap write – registers 40201-40203 (addressed 200 – 202)
08	Diagnostics – 00 – Return Query Data 01 – Restart Comm Option 02 – Return Diagnostic Register 04 – Force Listen Only Mode 10 – Clear Ctrs & Diag. Register 11 – Return Bus Message Count	basic diagnostics
16	Preset Multiple Registers	same as 06 - Preset Single Register
17	Report Slave ID	slave identification

The data structure access is available by reading/writing from/to appropriate registers (see table above). Every structure corresponds with appropriate continuous group of registers.

Example :

Controller state reading (NovarStatus) through „Read Input Registers“ function, address 1 supposed :

Command :

| 0x01| 0x04 | 0x00 | 0xC8| 0x00 | 0x1E | CRCLo | CRCHi |

NovarStatus is placed starting the input-register no. 30201(address 200 = 0xC8), structure length is 60 bytes = 30 registers (= 0x1E = 60 bytes).

Response :

| 0x01| 0x04 | 0x3C | register 30201 Hi| reg. 30201 Lo| reg. 30202 Hi | reg. 30202 Lo |
..... | reg. 30230 Hi | reg. 30230 Lo | CRCLo | CRCHi |

Behind address (0x01) , function number (0x04) and byte count (0x3C = 60) the individual registers follow. The register map below shows the structure layout in the registers.

The „NovarStatus” Structure Register Map

Address	Register	High Byte	LowByte
200	30201	SoftVersion H	SoftVersion L
201	30202	DeviceNo H	DeviceNo L
202	30203	DeviceType H	DeviceType L
203	30204	MTP H	MTP L
204	30205	Fr	I H
205	30206	I L	I50 H
206	30207	I50 L	Ir H
207	30208	Ir L	li H
208	30209	li L	Fi H
209	30210	Fi L	Kos
210	30211	THDU	THDI
211	30212	HarU3	HarU5
212	30213	HarU7	HarU9
213	30214	HarU11	HarU13
214	30215	HarU15	HarU17
215	30216	HarU19	HarI3
216	30217	HarI5	HarI7
217	30218	HarI9	HarI11
218	30219	HarI13	HarI15
219	30220	HarI17	HarI19
220	30221	U H	U L
221	30222	U50 H	U50 L
222	30223	CHL	DeltaI H
223	30224	DeltaI L	T
224	30225	Input	Res0
225	30226	MTN	Unom
226	30227	ActRelayState H	ActRelayState L
227	30228	Res1	Res2
228	30229	RegState	StateLEDs
229	30230	RegTime	ConfigChangeCnt

H=high byte, L=low byte

For the instrument parameters' setting, use the Config structure. It is placed starting the holding-register no. 40101(address 100), structure length is 100 bytes = 50 registers.

The „Config“ Structure Register Map

Address	Register	High Byte	LowByte
100	40101	RegMode	Res0
101	40102	ReqCos-0	SwitchDelayL-0
102	40103	SwitchDelayC-0	ReqCosBandWidth-0
103	40104	Res1-0	ReqCos-1
104	40105	SwitchDelayL-1	SwitchDelayC-1
105	40106	ReqCosBandWidth-1	Res1-1
106	40107	MTP H	MTP L
107	40108	SwitchBlockDelay	UIMode
108	40109	CSRatio	Ck
109	40110	Steps	QuickSteps
110	40111	CLVal0 H	CLVal0 L
111	40112	CLVal1 H	CLVal1 L
112	40113	CLVal2 H	CLVal2 L
113	40114	CLVal3 H	CLVal3 L
114	40115	CLVal4 H	CLVal4 L
115	40116	CLVal5 H	CLVal5 L
116	40117	CLVal6 H	CLVal6 L
117	40118	CLVal7 H	CLVal7 L
118	40119	CLVal8 H	CLVal8 L
119	40120	CLVal9 H	CLVal9 L
120	40121	CLVal10 H	CLVal10 L
121	40122	CLVal11 H	CLVal11 L
122	40123	CLVal12 H	CLVal12 L
123	40124	CLVal13 H	CLVal13 L
124	40125	FixedSteps H	FixedSteps L
125	40126	FixedStepValue H	FixedStepValue L
126	40127	LCosMargin	QuickControlSpeed
127	40128	AlarmSig H	AlarmSig L
128	40129	AlarmAction H	AlarmAction L
129	40130	FixedStepsFH	MTN
130	40131	Unom	TFHLimit(Fan)
131	40132	TFHLimit(Heating)	Ulimit(Undervoltage)
132	40133	Ulimit(Overvoltage)	THDUlimit
133	40134	THDlimit	CHLlimit
134	40135	Tlimit	SwitchNoLimit
135	40136	TCF	ScanFreq
136	40137	Res3	Res4
137	40138	DeviceAddr	RemoteBdRate
138	40139	AvePQWindowLength	Res5
139	40140	RemoteControl	ExtCosValue0
140	40141	ExtCosValue1	ExtCosValue2
141	40142	ExtCosValue3	ExtCosValue4
142	40143	Res6[0]	Res6[1]
143	40144	Res6[2]	Res6[3]

144	40145	OffsetCLVal-0 H	OffsetCLVal-0 L
145	40146	OffsetCLVal-1 H	OffsetCLVal-1 L
146	40147	OffsetMode	RemoteControlTimeout
147	40148	Res7[0]	Res7[1]
148	40149	Res7[2]	Res7[3]
149	40150	ConfigCRC H	ConfigCRC L

H=high byte, L=low byte

You can find other details about the instrument status in the Status and the EEStatus structures. They are placed one behind another, starting the input –register no. 30101(address 100), total length is 144 bytes = 72 registers.

The „Status” a the „EEStatus” Structure Register Map

Address	Register	High Byte	LowByte
100	30101	HWError	OutputSwitchNo0
101	30102	OutputSwitchNo1	OutputSwitchNo2
102	30103	OutputSwitchNo3	OutputSwitchNo4
103	30104	OutputSwitchNo5	OutputSwitchNo6
104	30105	OutputSwitchNo7	OutputSwitchNo8
105	30106	OutputSwitchNo9	OutputSwitchNo10
106	30107	OutputSwitchNo11	OutputSwitchNo12
107	30108	OutputSwitchNo13	Event H
108	30109	Event L	ActRelayState H
109	30110	ActRelayState L	ReqRelayState H
110	30111	ReqRelayState L	State
111	30112	AlarmSigActive H	AlarmSigActive L
112	30113	AlarmActionActive H	AlarmActionActive L
113	30114	BadSteps H	BadSteps L
114	30115	SoftVersion H	SoftVersion L
115	30116	DeviceNo H	DeviceNo L
116	30117	DeviceType H	DeviceType L
117	30118	PrecisedSteps H	PrecisedSteps L
118	30119	MaxTHDU	MaxTHDI
119	30120	MaxCHL	MaxHar3U
120	30121	MaxHar5U	MaxHar7U
121	30122	MaxHar9U	MaxHar11U
122	30123	MaxHar13U	MaxHar15U
123	30124	MaxHar17U	MaxHar19U
124	30125	Res0	Res1
125	30126	MaxT	MinCos
126	30127	MaxAveP H	MaxAveP L
127	30128	MaxAveQ H	MaxAveQ L
128	30129	MaxAveDeltaQ H	MaxAveDeltaQ L
129	30130	AveP0 HH	AveP0 H
130	30131	AveP0 L	AveP0 LL
131	30132	AveP1 HH	AveP1 H
132	30133	AveP1 L	AveP1 LL
133	30134	AveQ0 HH	AveQ0 H

134	30135	AveQ0 L	AveQ0 LL
135	30136	AveQ1 HH	AveQ1 H
136	30137	AveQ1 L	AveQ1 LL
137	30138	AveDeltaQ HH	AveDeltaQ H
138	30139	AveDeltaQ L	AveDeltaQ0 LL
139	30140	AvePQCounter0 HH	AvePQCounter0 H
140	30141	AvePQCounter0 L	AvePQCounter0 LL
141	30142	AvePQCounter1 HH	AvePQCounter1 H
142	30143	AvePQCounter1 L	AvePQCounter1 LL
143	30144	OutputSwitchNo64_0 H	OutputSwitchNo64_0 L
144	30145	OutputSwitchNo64_1 H	OutputSwitchNo64_1 L
145	30146	OutputSwitchNo64_2 H	OutputSwitchNo64_2 L
146	30147	OutputSwitchNo64_3 H	OutputSwitchNo64_3 L
147	30148	OutputSwitchNo64_4 H	OutputSwitchNo64_4 L
148	30149	OutputSwitchNo64_5 H	OutputSwitchNo64_5 L
149	30150	OutputSwitchNo64_6 H	OutputSwitchNo64_6 L
150	30151	OutputSwitchNo64_7 H	OutputSwitchNo64_7 L
151	30152	OutputSwitchNo64_8 H	OutputSwitchNo64_8 L
152	30153	OutputSwitchNo64_9 H	OutputSwitchNo64_9 L
153	30154	OutputSwitchNo64_10 H	OutputSwitchNo64_10 L
154	30155	OutputSwitchNo64_11 H	OutputSwitchNo64_11 L
155	30156	OutputSwitchNo64_12 H	OutputSwitchNo64_12 L
156	30157	OutputSwitchNo64_13 H	OutputSwitchNo64_13 L
157	30158	OutputSwitchOnTime2H_0 H	OutputSwitchOnTime2H_0 L
158	30159	OutputSwitchOnTime2H_1 H	OutputSwitchOnTime2H_1 L
159	30160	OutputSwitchOnTime2H_2 H	OutputSwitchOnTime2H_2 L
160	30161	OutputSwitchOnTime2H_3 H	OutputSwitchOnTime2H_3 L
161	30162	OutputSwitchOnTime2H_4 H	OutputSwitchOnTime2H_4 L
162	30163	OutputSwitchOnTime2H_5 H	OutputSwitchOnTime2H_5 L
163	30164	OutputSwitchOnTime2H_6 H	OutputSwitchOnTime2H_6 L
164	30165	OutputSwitchOnTime2H_7 H	OutputSwitchOnTime2H_7 L
165	30166	OutputSwitchOnTime2H_8 H	OutputSwitchOnTime2H_8 L
166	30167	OutputSwitchOnTime2H_9 H	OutputSwitchOnTime2H_9 L
167	30168	OutputSwitchOnTime2H_10 H	OutputSwitchOnTime2H_10 L
168	30169	OutputSwitchOnTime2H_11 H	OutputSwitchOnTime2H_11 L
169	30170	OutputSwitchOnTime2H_12 H	OutputSwitchOnTime2H_12 L
170	30171	OutputSwitchOnTime2H_13 H	OutputSwitchOnTime2H_13 L
171	30172	ManualStepValue H	ManualStepValue L

1.2.3 No-Parity Modbus-RTU Communication Frequent Problem

By definition, the Modbus-RTU protocol uses fundamentally nine-bit word length, supplemented with one start bit and one stop bit. When odd or even parity is set the ninth data carries information about the parity.

When no parity is set this bit is meaningless, but it must remain physically in transmitted words, otherwise instruments evaluate protocol error and do not accept the received message.

If an application program can not for some reason use nine-bit word transmission, you can work around this problem simply by setting two stop bits instead of one in the application program – then the instrument interprets the first stop as the parity bit, the protocol is correct and the received message is appropriately processed and answered.

1.2.4 Read Data Evaluation Example

Actual measured data and the instrument state are available in the "NovarStatus" data structure. The structure is mapped to the Input-registers starting from address 200, 30 registers in all. Reading of the structure looks as follows:

Request: 6.3.2013 16:10:13.64664 (+0.9063 seconds)

```
01 04 00 C8 00 1E F1 FC
```

Answer: 6.3.2013 16:10:13.66164 (+0.0156 seconds)

```
01 04 3C 00 15 FF FF 00 16 80 0A 4E 00 F5 00 8E
00 41 00 7E 00 3F 2E 04 89 06 0C 0E 06 06 00 01
00 00 D4 CF C8 A0 7A 69 65 67 5C 0A 0E 0A 19 AC
FF DA 1A 00 00 16 14 02 08 02 08 06 80 64 00 98
1B
```

1.2.4.1 Power Factor (Kos variable)

It is placed in the low byte of the register 209, value 0x2E = 46(dec), which corresponds to power factor value of 0.46 L.

1.2.4.2 Active (I_r) and Reactive (I_i) Current of the Fundamental Harmonic Component

Because of the instrument gets secondary current values only, you need the CT ratio (MTP variable) to get primary currents. The variable is in the register on address 203, its value is 0x800A; therefore, the CT ratio is A(hex)=10(dec). As the most significant bit is 1 the secondary nominal current is 5 A. Because of the ratio value is 10 the CT rating is 50/5 A.

Active and reactive current values are mapped to the register addresses 206 ÷ 208 – the high byte of the I_r is in the low byte of the register an address 206, i.e. 00; the low byte is in the high byte of the following register, i.e. 0x41; the complete value is 0x0041= 65(dec). Because of the current values are expressed in the 0.25 mA units, the I_r= 65 * 0.00025 =0.01625 A. This is the CT secondary current value, therefore it needs to be multiplied by the CT ratio to get the primary value : 0.01625 * 10 = 0.1625 A.

Similarly, the reactive current value 0x007E after recalculation gets 0.315 A of primary current.

1.2.4.3 Effective Current (I)

It is mapped to the registers on addresses 204 and 205 – the high byte is in the low byte of the register on address 204, i.e. 00; the low byte is in the high byte of the following register, i.e. 0xF5; complete value of 0x00F5 corresponds (recalculated in the same way as in the previous example) to 0.6125 A.

1.2.4.4 Effective Voltage (U) and Fundamental Harmonic Voltage (U₅₀)

If measured voltage connected to the instrument via a voltage transformer, it is necessary to find its ratio (MTN variable), because the instrument gets always secondary values only.

The VT ratio is in the high byte of the register 225, i.e. 0x16=22(dec). According the specified coding method (fractional function) the value is 220. In the low byte, the VT secondary nominal voltage is coded : 0x14=20(dec), that corresponds to 100 V. Finally, the VT rating is 22000/100 V.

The fundamental harmonic voltage (U_{50}) is in the register 221, i.e. $0x0A19=2585(\text{dec})$. Voltages are expressed in the 0.1 V units, so the secondary value is 258.5V. After multiplication by the VT ratio the primary values is $258.5 * 220 = 56,87 \text{ kV}$.

Similarly you can calculate effective voltage (U), placed in the register 220 (value $0x0A0E$).

1.2.4.5 Three Phase Active and Reactive Fundamental Harmonic Powers

Values of power are not transferred in the structure directly. It is necessary to calculate them from current and voltage values explained above.

To get a single phase active power, you need to multiply an active current by a fundamental harmonic voltage. *The UIMode* value, explained further below, defines if the voltage is *the phase* (line-to-neutral) type or *the line* (line-to-line) type.

In our example we suppose that the line type voltage is connected. Therefore, single phase active power is equal to product of active current and phase voltage. The phase voltage is the line voltage divided by square root of 3, i.e. $56.87 / 1.73 = 32.87 \text{ kV}$. This multiplied with the active current 0.1625 A gets 5,34 kW. Similarly, the single phase reactive power is 10.35 kvar.

If the voltage connected is phase type, the division by 1.73 is not performed.

Finally, multiply the phase powers by 3 to get three phase powers : $P_{ac} = 5.34 * 3 = 16.05 \text{ kW}$ and similarly the reactive power $P_{re} = 31.06 \text{ kvar}$.

Now how to determine if the measured voltage is phase type or line type. This information is contained in the *UIMode* variable in the „Config“ structure. The structure can be read as a sequence of *Holding-registers* starting from address 100 :

Request: 6.3.2013 17:40:22.11164 (+0.8750 seconds)

```
01 03 00 64 00 28 04 0B
```

Answer: 6.3.2013 17:40:22.14264 (+0.0312 seconds)

```
01 03 50 43 00 62 09 04 02 00 62 04 03 02 FF 80
0A 03 F5 00 01 0E FF 00 42 00 42 00 85 01 0A 02
15 02 15 02 15 02 15 02 15 02 15 02 15 02 15 02
15 02 15 FD F7 FD F7 7F 00 37 FF 32 FF 05 16 14
28 FB 50 6E 14 28 82 2D 64 01 FE FF FF 01 47 15
AB EE A1 DA 73
```

The UIMode variable is in the low byte of register 107, i.e. its value is $0xF5$, i.e. 1111 0101 binary. Bit No. 3 (numbered from 0) is 0, so the voltage the line type.

It is usually sufficient to read *the UIMode* value once when the program starts and then you can periodically read the „NovarStatus“ structure only, because the *UIMode* value does not change during the instrument operation.

For the program to be absolutely foolproof you can arbitrary check *the ConfigChangeCnt* variable in the „NovarStatus“ structure – its change means that the instrument setting was changed by an operator and therefore it is possible that *the UIMode* value was changed. Then one single-shot reading of this variable is necessary again.

1.2.5 Parameter change example – target power factor (ReqCos)

Let us assume we need to change target power factor (for tariff 1). It is placed in the "Config" structure under name *ReqCos[0]*.

You can find the structure description in the next chapter. It begins with the substructure "RegParType", when you see the ReqCos value coding. The Config structure itself begins with values RegMode and Res0. They can be accessed through the holding register on address 100.

At the next address 101, there is target power factor for tariff 1. It occupies 1 byte only and it is placed in the MSB bits of the holding register on address 101. It can be read as follows :

Request: 11.8.2014 11:24:22.82164 (+0.9844 seconds)

01 03 00 65 00 01 94 15

Answer: 11.8.2014 11:24:22.85264 (+0.0313 seconds)

01 03 02 62 09 51 22

Read value of the register is 6209 hexadecimally. 62hex corresponds to 98dec, so preset target power factor is 0.98.

The 09 is value of SwitchDelayL, i.e. control time for undercompensation. The value 09 corresponds (see the RegParType substructure description) to 3 minutes.

For the target power factor value change you can use, for example, the Preset Single Register function :

Request: 11.8.2014 11:32:43.49664 (+9.3906 seconds)

01 06 00 65 64 09 73 13

Answer: 11.8.2014 11:32:43.51264 (+0.0156 seconds)

01 06 00 65 64 09 73 13

As you can see, value 6409hex was written into the register on address 101 (=0065hex). As 64hex=100dec, the target power factor value is now 1.00.

It is not possible to write less than 1 register = 2 bytes in one operation. So when editing target power factor, that occupies one byte of the register only, you must firstly read whole register, then change its upper byte and then write whole register back.

Now, when reading the register back, you can check that the value is changed :

Request: 11.8.2014 11:33:13.65664 (+29.6406 seconds)

01 03 00 65 00 01 94 15

Answer: 11.8.2014 11:33:13.68764 (+0.0313 seconds)

01 03 02 64 09 52 82

1.3 Structures description

The syntax is conform to C-language. Long/ulong and int/uint variables are stored in high-low order (high byte is followed with low byte).

//=====

Status :

```
typedef struct {
    uchar    HWEError; /* hardware error info */
                /* bit0...EPROM error */
                /* bit1...RAM error */
                /* bit2...SEEPROM error */
                /* bit3...calibration error */
    uchar    OutputSwitchNo[14]; /* number of switch-ons, lower values */
    uint     Event; /* actual nonstandard event info */
                /* bit0...1=undercurrent */
                /* bit1...1=overcurrent */
                /* bit2...1=voltage loss */
                /* bit3...1=undervoltage */
                /* bit4...1=overvoltage */
                /* bit5...1=THDI exceeded */
                /* bit6...1=THDU exceeded */
                /* bit7...1=CHL exceeded */
                /* bit8...1=out of compensation */
                /* bit9...1=back feeding */
                /* bit10...1=switching number limit exceeded */
                /* bit11...1=step error */
                /* bit12...1=overheated */
                /* bit13...1=external alarm */
                /* bit14...1=connection unknown */
                /* bit15...1=step values unknown */
    uint     ActRelayState; /* actual relay state, 1=ON */
    uint     ReqRelayState; /* scheduled relay state */
    uchar    State; /* state */
                /* bit0-3...control process state */
                /* bit4...1= connection unknown */
                /* bit5...1= step values unknown */
    uint     AlarmSigActive; // active alarm signalling
    uint     AlarmActionActive; // active alarm action
                // both coded in the same way as „Event“
    uint     BadSteps; /* map of error steps (which are in standby mode) */
    uint     SoftVersion; /* lowbyte...software version */
                /* highbyte...special version */
    uint     DeviceNo; // serial number
    uint     DeviceType; // device type :
                /* 0x12...N1312 */
                /* 0x13...N1206 */
                /* 0x14...N1214 */
                /* 0x15...N1106 */
                /* 0x16...N1114 */
} SType; /* Status -34 bytes */
```

//=====

EEStatus :

```
typedef struct {
    uint     PrecisedSteps; /* precised steps map */
                /* 1=precised step, 0=not precised yet */
    uchar    MaxTHD[2]; /* max. value since last clearing */
                // THD[0]...THDU, THD[1]...THDI
                // THD coding :
                // 0 ÷100 ... 0.00 ÷ 50.0%, step 0,5%
```

```

//      101÷200 ... 52.5 ÷ 300.0%, step 2,5%
//      201÷250 ... 310 ÷ 800%, step 10%
//      255 ... undefined
uchar  MaxCHL;      /* max. value since last clearing */
// CHL coding :
//      0 ÷150 ... 0 ÷ 150%, step 1%
//      151÷200 ... 155 ÷ 400%, step 5%
//      201÷250 ... 410 ÷ 900%, step 10,0%
//      255 ... undefined
uchar  MaxHar[9];   /* max. value of 3-5-7-9-11-13-15-17-19th voltage harmonic */
// Har coding :
//      0 ÷100 ... 0.00 ÷ 10.0%, step 0,1%
//      101÷200 ... 10.5 ÷ 60.0%, step 0,5%
//      201÷254 ... 62.5 ÷ 195.0%, step 2,5%
//      255 ... undefined
uchar  Res0;        /* res */
uchar  Res1;        /* res */
char   MaxT;        /* max. temperature since last clearing, i degree of Celsius*/
char   MinKos;      /* minimum cos value since last clearing */

int    MaxAveP;     // max. value of Pavg
int    MaxAveQ;     // max. value of Qavg
int    MaxAveDeltaQ; // max. value of DeltaQavg
// coded as current, to be multiplied with Unom to get power values

float  AveP[2];     // sliding average of P (for internal use only)
float  AveQ[2];     // sliding average of Q (for internal use only)
float  AveDeltaQ;   // sliding average of DeltaQ (for internal use only)
ulong  AvePQCounter[2]; // sliding window counter (for internal use only)

uint   OutputSwitchNo64[14]; // number of step switching in 64-units („higher“ values )
// OutputSwitchNo need to be added to get
// real number of switching operations
uint   OutputSwitchOnTime2H[14]; /* switch-on time of each step */
// in 2-hour units */
// max. range 65000, i.e. 130000 hours */
uint   ManualStepValue; /* step map in manual mode; 0=ON, 1=OFF */
} EESType; /* EEStatus-110 bytu */

```

```
//=====
```

NovarStatus :

```

typedef struct {
uint   SoftVersion; /* low byte...software version */
// high byte...special version number(usually zero) */
uint   DeviceNo;    /* serial number */
uint   DeviceType;  /* device type, the same as in the Status */
uint   MTP;         /* CT ratio, 1-6000 */
// bits 14-0...CT primary nominal value in 5A-units */
// bit 15...0= CT secondary nom. value 1A (CT-ratio xxx/1A) */
// 1= CT secondary nom. value 5A (CT-ratio xxx/5A) */
uchar  Fr;          // frequency; coding :
// step 0.1Hz, 55Hz=0x80, 0= 42,2Hz, 254=67.6Hz, 255=undefined
uint   I;           /* eff. current (on secondary side of the CT) in 0,25mA units*/
// as all current values, the Ck and step values, it must be */
// multiplied with the CT-ratio to get primary values */
uint   I50;        /* fundamental harmonic current value in 0,25mA-units*/
int    Ir;         /* fund. harmonic active component current value in 0,25mA-units*/
int    Ii;         /* fund. harmonic reactive component current value in 0,25mA-units*/
// both signed, negative means export of active energy or */
// capacitive character of reactive energy */
int    Fi;         // voltage vs. current phasor angle in degrees
char   Kos;        /* cos fi in 0,01-units, negative=capacitive character */
// if(Kos == 0) -> cos fi = 0.00L */

```

```

/* . . . */
/* if(Kos == 99) -> cos fi = 0.99L */
/* if(Kos == 100) -> cos fi = 1.00 */
/* if(Kos == -99) -> cos fi = 0.99C */
/* . . . */
/* if(Kos == -1) -> cos fi = 0.01C */
/* if(Kos == -100) -> cos fi = 0.00C */
/* . . . */
/* if(Kos == 127)-> cos fi undefined (for ex. when I==0) */
uchar THD[2]; // 0...U, 1...I; for coding see EEStatus
uchar Har[2][9]; // 0...U, 1...I, value of 3-5-7-9-11-13-15-17-19th harmonic
// for coding see EEStatus
uint U; // effective voltage, in 0.1V steps; 0xFFFF...undefined
uint U50; // fundamental harmonic of voltage U, in 0.1V steps; 0xFFFF...undefined
uchar CHL; // Capacitor Harmonic Load factor; for coding see EEStatus */
int Deltai; // missing fund. harmonic reactive component current; for coding see I
char T; // temperature in degrees of Celsius
uchar Input; // bit 0: ...0 = external input open
// ...1 = external input closed
uchar Res0; /* reserve */

uchar MTN; // VT-ratio
// coding :
// 0=1(without VT)
// 1=10 ÷ 100=1000, i.e. 1000/100 ÷ 100000/100
// 101=1100 ÷ 140=5000, i.e. 110000/100 ÷ 500000/100
// 140...500kV/100V, maximum
// >140...withot VT, direct conection, the same meaning as 0
uchar Unom; // nominal measurement voltage (without VT-ratio)
// coding :
// 9 ...50V
// 10 ...55V
// 11 ...58V
// 12 ÷ 150 ...60V ÷ 750V, step 5 V

uint ActRelayState; /* actual relay state, 1=ON */
uchar Res1;
uchar Res2;
uchar RegState; /* controller state */
// STATEMASK 0x0F /* lower 4 bits-controller state : */
/* specifies actual controller state */
// STATEINIT 0x00 /* after-reset state */
// STATETEST 0x01 /* start-up test in progress */
// STATEUIMODERE 0x02 /* UIMode-recognition ("AP") in progress */
// STATEUIMODEEUK 0x03 /* UIMode-unknown ("P=0") */
// STATECLVALUESRE 0x04 /* CLValues-recog. ("AC") in progress */
// STATECLVALUESUK 0x05 /* CLValues-unknown ("C=0") */
// STATERUN 0x06 /* normal control in progress */
// STATESTANDBYCLOFF 0x07 /* standby-outputs OFF (excl. fixed-steps) */
// STATESTANDBYALLOFF 0x08 /* standby - all outputs OFF */
// STATEIDLE 0x09 /* idle - no measured data available */
// STATEMANUAL 0x0F /* manual mode */
/* upper 4 bits – non-standard states mask */
// STATEUIMODEUNKNOWN 0x10 /* UIMode unknown ("P=0") */
// STATECLVALUESUNKNOWN 0x20 /* CLValues unknown ("C=0") */
// STATEVOLTAGEBAD 0x40 /* voltage too low („U=0“) */
// STATECURRENTLOW 0x80 /* current too low („I=0“) */
uchar StateLEDs; /* panel indicating LED-diodes state */
/* bit0...TrendL */
/* bit1...TrendLFlash */
/* bit2...TrendC */
/* bit3...TrendCFlash */
/* bit4...PwrReverse */
/* bit5...Alarm */
/* bit6...reserve */

```

```

        /* bit7...Error */
        uchar   RegTime; /* time to next control action in %; it decreases from 100 to 0 */
        uchar   ConfigChangeCnt;
    } NSType; /* NovarStatus-60 bytes */
//=====

```

Config :

```

typedef struct {
    char   ReqCos; /* target cos, range from -80 to +80 */
                /* 0x7F... value unknown */
                /* or in degrees: 101= +10° ÷ 121= -10° */
    uchar   SwitchDelayL; /* control period for undercompensation */
    uchar   SwitchDelayC; /* control period for overcompensation */
                /* control periods coding, bits 3÷0: */
                /* 0...5 sec */
                /* 1...10 sec */
                /* 2...15 sec */
                /* 3...20 sec */
                /* 4...30 sec */
                /* 5...45 sec */
                /* 6... 1 min */
                /* 7... 1 min 30 sec */
                /* 8... 2 min */
                /* 9... 3 min */
                /* 10... 4 min */
                /* 11... 5 min */
                /* 12... 7 min */
                /* 13... 10 min */
                /* 14... 15 min */
                /* 15... 20 min */
                /* otherwise...value not valid */
                /* bit 7...0= square-proportional control time */
                /* ...1= linear-proportional control time */
    uchar   ReqCosBandWidth; // control bandwidth width
                // in 0.005 steps, range 0÷8, i.e. 0.000 ÷ 0.040
    char   Res1; /* res.*/
} RegParType;

typedef struct {
    uchar   RegMode; /* control mode setting */
                /* bit0...0=manual, 1=automat */
                /* bit1...0=evaluation of tarif2 input */
                /* bit2...1=automatic step value recognition active */
                /* bit3...1= password not retained- required*/
                /* 0= password retained - not required*/
                // bit 4...if bit1 is active(0) :
                // ...1(default)...tariff2 activated by 2nd-tariff input
                // ...0.....tariff2 activated by back feeding event
                /* bit5...1...if bit 2 active(1)...AUTO-mode of step recognition
                // bit 6...= 1...standard control
                // = 0...linear control mode, for harmonic filters control
                /* bit7...res. */
    uchar   Res0; /* no sense */
    RegParType RegPar[2]; /* second set of values valid for 2nd tariff */
    uint   MTP; /* CT ratio 1-9950 */
                /* bits 14-0...primary value in 5A-units */
                /* bit 15...0= CT- nominal secondary current 1A */
                /* 1= 5A */
    uchar   SwitchBlockDelay; /* reconnection block time */
                /* for coding see SwitchDelayL/C */
    uchar   UIMode; /* connection type */
                /* bits 2-0: */
                /* for bit 3=1, i.e. phase voltage : */
                /* 1...U10 */

```

```

/*      2...U20      */
/*      3...U30      */
/*      4...U01      */
/*      5...U02      */
/*      6...U03      */
/* for bit 3=0, i.e. line voltage : */
/*      1...U12      */
/*      2...U23      */
/*      3...U31      */
/*      4..U21      */
/*      5..U32      */
/*      6..U13      */
/* bits 7-4 : */
/* if bits 0-3 out of range: */
/* upper nibble=0.....aut. recognition unsuccessful */
/* upper nibble=1-F...UIMode not set yet, automatic */
/* recognition process will be started */
uchar  CSRatio; /* compensation step ratio */
/*      0x00...-individual settings */
/*      1...1:1:1:1:1 */
/*      2...1:1:2:2:2 */
/*      3...1:1:2:2:4 */
/*      4...1:1:2:3:3 */
/*      5...1:1:2:4:4 */
/*      6...1:1:2:4:8 */
/*      7...1:2:2:2:2 */
/*      8...1:2:3:3:3 */
/*      9...1:2:3:4:4 */
/*     10...1:2:3:6:6 */
/*     11...1:2:4:4:4 */
/*     12...1:2:4:8:8 */
uchar  Ck; /* from 0,02 to 2A in 0,01A-steps, coded in the same */
/* way as current */
uchar  Steps; /* number of inductive and capacitive steps used */
/* bits 3-0...CSteps; bits 7-4...LSteps */
uchar  QuickSteps; /* for Novar1312 only: number of transistor group */
/* steps used; for other types without sense */
int    CLVal[14]; /* step values*/
/* coded in the same way as current (see NovarStatus) */
/* capacitors-positive value, inductor-negative value */
/* value 0x7FFF...the step value is unknown */
uint   FixedSteps; /* bit map of fixed steps, 0=fixed step */
uint   FixedStepValue; /* step value of fixed steps; 0=ON, 1=OFF */
char   LCosMargin; /* limit cos-value for decompensation choke operation */
/* coded in the same way as the cos fi (see NovarStatus) */
uchar  QuickControlSpeed; /* for Novar1312 only: transistor group control */
/* speed and reconnection block time */
//      value  contr./sec  block. time [s]
//      0      1          10 (default)
//      1      1          5.0
//      2      1          2.0
//      3      1          1.0
//      4      2          5.0
//      5      2          2.5
//      6      2          1.0
//      7      2          0.5
//      8      3          3.3
//      9      3          1.7
//     10      3          0.7
//     11      3          0.3
//     12      5          2.0
//     13      5          1.0
//     14      5          0.4
//     15      5          0.2
//     16     10          1.0

```



```

//      17      10      0.5
//      18      10      0.2
//      19      10      0.1
uint   AlarmSig;    /* alarm signalling map */
// for coding see Event
uint   AlarmAction; /* alarm action map, similar to AlarmSig */
uchar  FixedStepsFH; /* optional function of last two steps; if the step as fixed :
// bit 1,0...for last step :
// bit 0...1=optional function off
//      0= optional function on; in such case :
// bit 1...1=F(fan)
//      0=H(heating)
// bit 3,2...the same meaning for step before the last
uchar  MTN;        /* CT-ratio; for coding see NovarStatus
uchar  Unom;       /* nominal measurement voltage; for coding see NovarStatus
char   TFHLimit[2]; /* 0...temperature limit for fan switching
// 1...temperature limit for heating switching, in deg. of Celsius
uchar  ULimit[2];  /* voltage limit in percents of Unom:
// 0... for undervoltage, 1...for overvoltage
// range 10% ÷ 150%
uchar  THDLimit[2]; /* THD limit, 0...for voltage, 1...for current
// coding as THD; 0xFF.... off
uchar  CHLLimit;   /* CHL limit, coding as CHL
uchar  TLimit;     /* temperature limit, coding as T
uchar  SwitchNoLimit; /* switch number limit value for alarm signalling */
// * from 10000 do 2000000, in 10000-units*/
uchar  TCF;       /* temp. display : bit 0...1=Celsius, 0=Fahrenheit
uchar  ScanFreq;  /* bity 1, 0 :
//      1 x      auto
//      0 1      fixed 50 Hz
//      0 0      fixed 60 Hz
uchar  Res3;      /* res */
uchar  Res4;      /* res */
uchar  DeviceAddr; /* address */
uchar  RemoteBdRate; /* low nibble = Bd-rate */
// * 6..4800 Bd */
// * 7..9600 Bd */
// * 8..19200 Bd */
// * high-nibble = protocol: */
// * bit7...reserve (0) */
// * bit6...0 = protocol KMB */
// * 1 = ModBus RTU */
// * bits 5,4...parity (for ModBus only) */
// * bit5.....0=no parity(i.e. 2stopbits)*/
// * 1=parity: */
// * bit4.....0=even */
// * 1=odd */
uchar  AvePQWindowLength; /* sliding window legths for P/Q/cos averaging and
// minimum/maximum values evaluation
// low nibble...for avg, high nibble...for max/min
// coding:
//      0...1min.....1/12
//      1...15min.....1/180
//      2...1hour.....1/720
//      3...8hours.....1/5760
//      4...1day.....1/17280
//      5...7days.....1/120960
//      otherwise...7 days
uchar  Res1;      /* res

// following 20 bytes for firmware version 1.3 and higher :

uchar  RemoteControl; /* for NRC version only, irrelevant for Modbus access
char   ExtCosValue[5]; /* the same
uchar  Res6[4];      /* reserve

```

```

int      OffsetCLVal[2];          // offset power values for tariff 1 and 2
                                     // data format equal to the CLVal
uchar    OffsetMode;             // bit0 = 1 ... standard control, without offset power
                                     //      0 ... control with offset power
uchar    RemoteControlTimeout;   // for NRC version only, irrelevant for Modbus access
uchar    Res7[4];                // reserve
    
```

```

uint     ConfigCRC;              /* no sense, may be set to any value */
} CType;                          /* Config - 100 bytes (80 bytes for firmware version 1.2 and older) */
    
```

//===

NovarSetMap :

```

typedef struct {
uchar ClearLimit;                /* maximum values clearing and mode setting (value 1 active) */
                                     // maximum values clearing :
                                     // bit0= Acos, APac, Apre
                                     // bit1= minCos,maxPac,maxPre,maxdPre
                                     // bit2= MaxTemp
                                     // bit3=maxCHL,maxTHDU,maxharU
                                     // bit4=maxTHDI
uint ClearSwitchNo;             // bit0-13... clear switching number of appropriate step
uchar Switch;                   // controller action command
                                     // bit0=lock editation(password will be required)
                                     // bit1=go to control mode(if controller in manual mode)
                                     // bit2=controller reinitialisation
                                     // bit3=clear HWError-info
uint ClearSwitchOnTime;         // bit0-13... clear switch-on time of appropriate step
} NovarSetMapType; /* NovarSetMap -6 bytes */
    
```

//===